PERANCANGAN IP*CORE* MIKROKONTROLER KOMPATIBEL ATMEL ATMEGA8535 DENGAN VHDL

TESIS

Oleh

DIDI SURIAN
06 06 00 3291

PROGRAM STUDI TEKNIK ELEKTRO
PROGRAM PASCASARJANA BIDANG ILMU TEKNIK
UNIVERSITAS INDONESIA
GANJIL 2007/2008

PERANCANGAN IP*CORE* MIKROKONTROLER KOMPATIBEL ATMEL ATMEGA8535 DENGAN VHDL

TESIS

Oleh

DIDI SURIAN
06 06 00 3291

TESIS INI DIAJUKAN UNTUK MELENGKAPI SEBAGIAN PERSYARATAN MENJADI MAGISTER TEKNIK

PROGRAM STUDI TEKNIK ELEKTRO
PROGRAM PASCASARJANA BIDANG ILMU TEKNIK
UNIVERSITAS INDONESIA
GANJIL 2007/2008

PERNYATAAN KEASLIAN TESIS

Saya menyatakan dengan sesungguhnya bahwa tesis dengan judul :

PERANCANGAN IP*CORE* MIKROKONTROLER KOMPATIBEL ATMEL ATMEGA8535 DENGAN VHDL

yang dibuat untuk melengkapi sebagian persyaratan menjadi Magister Teknik pada Kekhususan Desain VLSI Program Studi Teknik Elektro Program Pascasarjana Universitas Indonesia, sejauh yang saya ketahui bukan merupakan tiruan atau duplikasi dari tesis yang sudah dipublikasikan dan atau pernah dipakai untuk mendapatkan gelar kesarjanaan di lingkungan Universitas Indonesia maupun di Perguruan Tinggi atau Instansi manapun, kecuali bagian yang sumber informasinya dicantumkan sebagaimana mestinya.

Depok, 14 Desember 2007

Didi Surian
NPM 0606003291

PENGESAHAN

Tesis dengan judul :

PERANCANGAN IP*CORE* MIKROKONTROLER KOMPATIBEL ATMEL ATMEGA8535 DENGAN VHDL

dibuat untuk melengkapi sebagian persyaratan menjadi Magister Teknik pada Kekhususan Desain VLSI Program Studi Teknik Elektro Departemen Teknik Elektro Fakultas Teknik Universitas Indonesia. Tesis ini telah diujikan pada sidang ujian tesis pada tanggal 2 Januari 2008 dan dinyatakan memenuhi syarat/sah sebagai tesis pada Departemen Teknik Elektro Fakultas Teknik Universitas Indonesia.

Depok, 2 Januari 2008

Dosen Pembimbing

Prof. Dr-Ing. Ir. Harry Sudibyo S., DEA

NIP 130891668

UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada :

Prof. Dr-Ing. Ir. Harry Sudibyo Soejoktro, DEA

selaku dosen pembimbing yang telah bersedia meluangkan waktu untuk memberi pengarahan, diskusi, dan bimbingan serta persetujuan sehingga laporan tesis ini dapat selesai dengan baik.

Didi Surian
NPM 06 06 00 3291
Departemen Teknik Elektro

Dosen Pembimbing
Prof. Dr-Ing. Ir. Harry Sudibyo Soejoktro, DEA

PERANCANGAN IP*CORE* MIKROKONTROLER KOMPATIBEL ATMEL ATMEGA8535 DENGAN VHDL

ABSTRAK

Perancangan suatu sistem dijital dengan konsep penggunaan rancangan kembali dapat meningkatkan produktivitas dari perancang. Melalui penggunaan konsep perancangan dengan menggunakan IP (*Intellectual Property*) *core*, seorang perancang dapat melakukan verifikasi dan simulasi terhadap rancangan yang telah dibuatnya. Dengan demikian dapat meminimalkan kesalahan yang terjadi sebelum proses produksi dilakukan. Salah satu sistem dijital yang telah banyak dipakai dalam dunia industri hingga mainan anak-anak adalah mikrokontroler. Mikrokontroler dapat dipandang sebagai sebuah komputer dengan beberapa modul pendukung seperti antarmuka *input output* dan *memory*.

Tesis ini memberikan kontribusi berupa IP*core* sebuah mikrokontroler yang kompatibel dengan mikrokontroler Atmel ATMega 8535. IP*core* yang dirancang memakai bahasa perangkat keras VHDL. Pengertian kompatibel disini adalah mikrokontroler yang dirancang dapat mengerti instruksi-instruksi seperti pada mikrokontroler Atmel ATMega 8535. Perancangan dimulai dari modul-modul di dalam mikrokontroler, simulasi modul-modul, dan terakhir mengintegrasikan semua modul untuk membentuk UIMega 8535. Simulasi dan verifikasi mencakup modul-modul berikut: ROM, *instruction register, timer interrupt, prescallerWDR* dan *core* UIMega 8535. Modul *core* UIMega 8535 terdiri dari unit *decode,* ALU, RAM, *status register, general purpose register, program counter, stack,* dan *external interrupt*. Simulasi dan verifikasi dilakukan dengan penyajian dalam bentuk diagram pewaktuan keluaran terhadap masukan yang diberikan.

Kata Kunci : Mikrokontroler, IP*Core*, VHDL

Didi Surian
NPM 06 06 00 3291
Electrical Engineering Department

Counselor
Prof. Dr-Ing. Ir. Harry Sudibyo Soejoktro, DEA.

DESIGN OF ATMEL ATMEGA8535 COMPATIBLE MICROCONTROLLER IP*CORE* WITH VHDL

ABSTRACT

A digital system design with reusability concept can increase the designers' productivity. By using IP (*Intellectual Property*) *core* design concept, designers can do verification and simulation to their design. So they can minimalize the failure factor before the production process. One of digital system which is most used in industry to toys is microcontroller. Microcontroller can be seen as a computer with several supporting modules, such as *input output* interfaces and *memory*.

This thesis contributes the microcontroller IP*core* which is compatible with Atmel ATMega 8535 microcontroller. The IP*core* is designed by using hardware description language VHDL. Compatible here means that designed microcontroller understands the Atmel ATMega 8535's instructions. The research designs microcontroller's modules, simulates the modules, and finally integrates all modules to build UIMega 8535. Simulation and verification cover the following modules: ROM, *instruction register, timer interrupt, prescallerWDR* and *core* UIMega 8535. The *core* UIMega 8535 consists of these units: *decode*, ALU, RAM, *status register*, *general purpose register, program counter, stack*, and *external interrupt*. Simulation and verification is done by presenting *output* timing diagram for given *input*.

Keywords : Microcontroller, IP*Core*, VHDL

DAFTAR ISI

Halaman

Halaman

DAFTAR GAMBAR

Halaman

Halaman

DAFTAR TABEL

Halaman

DAFTAR LAMPIRAN

Halaman

DAFTAR SINGKATAN

C	Carry Flag
Z	Zero Flag
N	Negative Flag
V	Two's complement Overflow indicator
S	N $\oplus$ V, Signed test
H	Half Carry Flag
T	Transfer bit
I	Global Interrupt Enable/Disable Flag
Rd	Destination Register di Register File
Rr	Source Register di Register File
K	Data konstan
k	Alamat konstan
b	Bit di Register File atau Register I/O (3-bit)
s	Bit di Status Register
X,Y,Z	Register pengalamatan tidak langsung (X=R27:26, Y=R29:28, Z=R31:R30)
A	Alamat lokasi I/O
q	Alamat perpindahan (displacement) untuk pengalamatan langsung (6-bit)
IC	Integrated Circuit
IP	Intellectual Property
RISC	Reduced Instructions Set Computer
VHSIC	Very High Speed Integrated Circuit
VHDL	VHSIC Hardware Description Language
CAD	Computer-Aided Design
MAX+plus	Multiple Array MatriX Programmable Logic User System
AHDL	Altera Hardware Description Language
ALU	Arithmetic Logic Unit
IEEE	Institute of Electrical and Electronics Engineers

TIMSK	Timer/Counter Interrupt Mask Register
TOIE0	Timer0/Counter0 Compare Match Interrupt Enable
TCCR0	8-bit Timer0/Counter0 Register Description
CS	Clock Select
TCNT0	Timer0/Counter0 Register
TIFR	Timer/Counter Interrupt Flag Register
TOV0	Timer0/Counter0 Overflow Flag
MCUCR	MCU Control Register
ISC	Interrupt Sense Control
GICR	General Interrupt Control Register
INT0	Interrupt Request 0

BAB I
PENDAHULUAN

1.1 LATAR BELAKANG

Perkembangan teknologi dijital telah menunjukkan pengaruh yang luar biasa bagi kehidupan manusia. Dimulai sejak kurang lebih era tahun 60-an dimana suatu rangkaian elektronik masih terdiri dari komponen-komponen yang sangat besar ukurannya hingga sekarang ini yang telah berkembang teknologi rangkaian terpadu. Suatu rangkaian terpadu atau umumnya disingkat menjadi IC (*Integrated Circuit*) dapat terdiri hingga dengan jutaan transistor per cm^2 satuan luas.

Perancangan suatu rangkaian dijital selalu berhubungan permasalahan jumlah rangkaian logika yang hendak diimplementasikan pada sebuah serpih (*chip*). Seorang perancang sistem dijital dapat saja menggunakan rangkaian logika yang telah tersedia sehingga tidak perlu merancang dari awal namun hal ini memiliki kelemahan, yaitu fungsionalitas dari rancangan yang telah ada bersifat tetap dan tidak dapat diubah. Perancangan dengan metoda penggunaan kembali (*reuse*) akan efektif untuk meningkatkan produktivitas [1].

Intellectual Property (IP) dalam bentuk fungsi-fungsi yang telah didefinisikan, atau disebut juga *core*, saat ini telah menjadi topik pembahasan yang hangat dalam dunia industri elektronik [2]. Perancangan dengan menggunakan IP *core* merupakan suatu metode perancangan dengan sifat penggunaan kembali sehingga selain dapat meningkatkan waktu perancangan juga dapat meningkatkan kualitas perancangan. Penggunaan IP *core* dalam perancangan rangkaian logika memberikan keuntungan bagi para desain sistem dijital karena sebelum diimplementasikan dapat dilakukan verifikasi lewat simulasi sehingga dapat meminimalkan kesalahan desain.

Salah satu penerapan IP *core* adalah dalam mewujudkan suatu mikrokontroler. Sebuah mikrokontroler (disebut juga MCU atau µC) diumpamakan sebagai sebuah komputer di dalam sebuah serpih [3]. Mikrokontroler umumnya dilengkapi dengan beberapa modul pendukung seperti

memory dan antar muka masukan dan keluaran (*I/O interface*). Penggunaan mikrokontroler dimulai dari peralatan automasi peralatan industri hingga pada mainan anak-anak.

Penelitian mengenai perancangan IP *core* suatu mikrokontroler akan memberikan suatu pemahaman dalam mengembangkan mikrokontroler yang lebih maju, baik dalam hal kecepatan maupun banyaknya instruksi yang dapat ditangani.

1.2 PERUMUSAN MASALAH

Pada tesis ini dilakukan perancangan mikrokontroler kompatibel dengan mikrokontroler ATMEL ATMega 8535. Pengertian kompatibel ini adalah mikrokontroler yang dirancang dapat mengerti kode-kode instruksi yang dimiliki oleh ATMega 8535. Hasil perancangan akan diberi nama UIMega 8535. Perancangan akan dilakukan modul per modul dan kemudian digabung untuk dilakukan verifikasi secara keseluruhan modul. Verifikasi akan dilakukan dengan simulasi pewaktuan.

Perancangan IP *core* mikrokontroler kompatibel Atmel ATMega 8535 akan dilakukan dengan menggunakan bahasa perangkat keras VHDL (*Very high-speed Hardware Description Language*).

1.3 TUJUAN PENELITIAN

Tesis ini bertujuan memberikan kontribusi berupa IP *core* mikrokontroler kompatibel dengan mikrokontroler ATMega 8535 yang merupakan salah satu mikrokontroler tipe RISC (*Reduced Instructions Set Computer*) keluaran Atmel. IP *core* UIMega 8535 dirancang untuk dapat mengeksekusi instruksi-instruksi aritmatik, logika, perpindahan data register, dan operasi bit yang dimiliki oleh ATMega 8535 hanya dalam satu sinyal *clock*, sehingga UIMega 8535 dapat lebih cepat dalam mengeksekusi instruksi tersebut.

IP *core* yang dihasilkan dalam bahasa HDL (*Hardware Description Language*) yaitu VHDL. Kontribusi ini diharapkan dapat menjadi acuan bagi para desainer sistem dijital dalam mengembangkan mikrokontroler dengan fitur-fitur lainnya seperti jumlah instruksi yang dapat ditangani, kecepatan pemrosesan yang lebih tinggi, dan lain sebagainya.

1.4 BATASAN MASALAH

Perancangan ini akan dibatasi dalam beberapa hal seperti berikut di bawah ini:

1. Jumlah register dalam *General Purposes Registers* dibatasi 16 buah *register* dari 32 buah *register*. Hal ini dikarenakan implementasi 32 register akan memakan tempat, sehingga sebagian besar divais target tidak dapat digunakan untuk modul lain.
2. Pengalamatan pada modul ROM hanya selebar 9-bit, yaitu sebanyak 512 lokasi dengan lebar 16-bit, sehingga total kapasitas ROM yang didukung adalah 1KBytes. Hal ini juga untuk efisiensi penggunaan divais target.
3. Desain memiliki fitur I/O dengan fungsi hanya sebagai digital I/O, yaitu *port* A dan B sebagai *input* sedangkan *port* C dan D sebagai *output* dan tidak memiliki EEPROM serta hanya memiliki satu buah *timer* yaitu *timer0*.
4. *Sleep mode* dan *break* hanya akan menghentikan mikrokontroler mengeksekusi perintah sampai terjadi *interrupt*.

1.5 METODOLOGI PENELITIAN

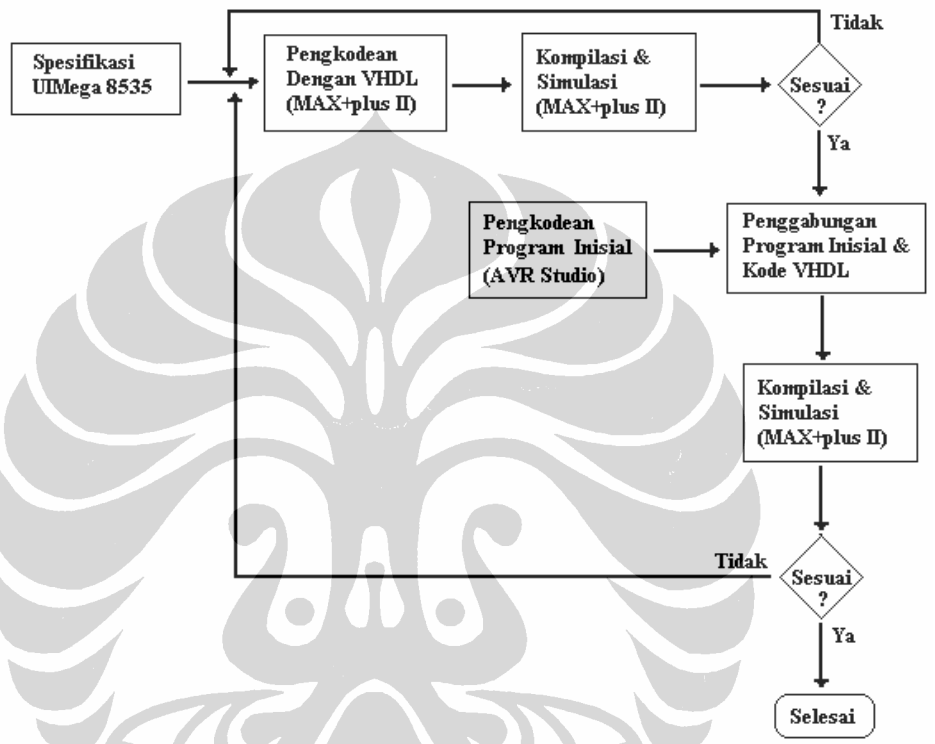
Tesis ini terdiri dari beberapa tahapan dan melibatkan beberapa aplikasi CAD (*Computer-Aided Design*) *tool* pendukung. Tahapan pertama adalah perancangan modul-modul dari mikrokontroler dengan bahasa VHDL. Aplikasi pendukung yang terlibat adalah aplikasi MAX+plus II (*Multiple Array MatriX Programmable Logic User System*) dari Altera. Aplikasi MAX+plus II adalah suatu aplikasi yang ditujukan untuk pendesainan rangkaian logika dengan berbagai metoda pemrograman. MAX+plus II dapat menerima dan memproses file bertipe AHDL (*Altera Hardware Description Language*), Verilog HDL, VHDL dan skematik OrCAD.

Setiap modul yang telah dirancang diverifikasi dengan menggunakan simulasi diagram pewaktuan. Verifikasi juga dilakukan setelah modul-modul yang ada diintegrasikan menjadi satu kesatuan. Tahapan ini akan dilakukan secara berulang hingga tidak terdapat kesalahan.

Tahapan selanjutnya adalah dengan memberikan program tertentu ke dalam mikrokontroler yang telah dirancang. Tahapan ini membutuhkan aplikasi

pendukung, yaitu Atmel AVR Studio yang sudah terintegrasi dengan AVR Assembler. Aplikasi ini akan menghasilkan format file *.hex* yang akan digunakan untuk mendefinisikan nilai awal dari komponen *memory* yang dihasilkan oleh aplikasi MAX+plus II.

Secara umum metodologi penelitian yang digunakan dalam tesis ini diperlihatkan pada Gambar 1.1 berikut ini.



Gambar 1.1 Metodologi penelitian

1.6 SISTEMATIKA PENULISAN

Laporan tesis ini secara keseluruhan berisi lima bab dengan perincian sebagai berikut. Bab I merupakan pendahuluan yang berisi latar belakang masalah, perumusan masalah, tujuan penelitian, batasan masalah, metodologi penelitian, dan sistematika penulisan. Bab II membahas mengenai landasan teori mengenai mikrokontroler Atmel ATMega 8535 yaitu mengenai arsitektur dan instruksi yang didukung. Pada bab ini juga dibahas mengenai bahasa perangkat keras yang dipakai yaitu VHDL. Dalam Bab III, dipaparkan mengenai perancangan IP *core* mikrokontroler UIMega 8535 yang dibahas modul per modul

beserta tahapan perancangan dengan menggunakan aplikasi MAX+plus II. Bab IV membahas mengenai hasil pengujian UIMega 8535 yang dilakukan modul per modul. Bab ini juga membahas pengujian terhadap instruksi yang dimiliki oleh ATMega 8535 pada UIMega 8535. Hasil pengujian akan ditampilkan dalam bentuk simulasi pewaktuan. Bab V memaparkan kesimpulan yang didapat dari tesis ini.

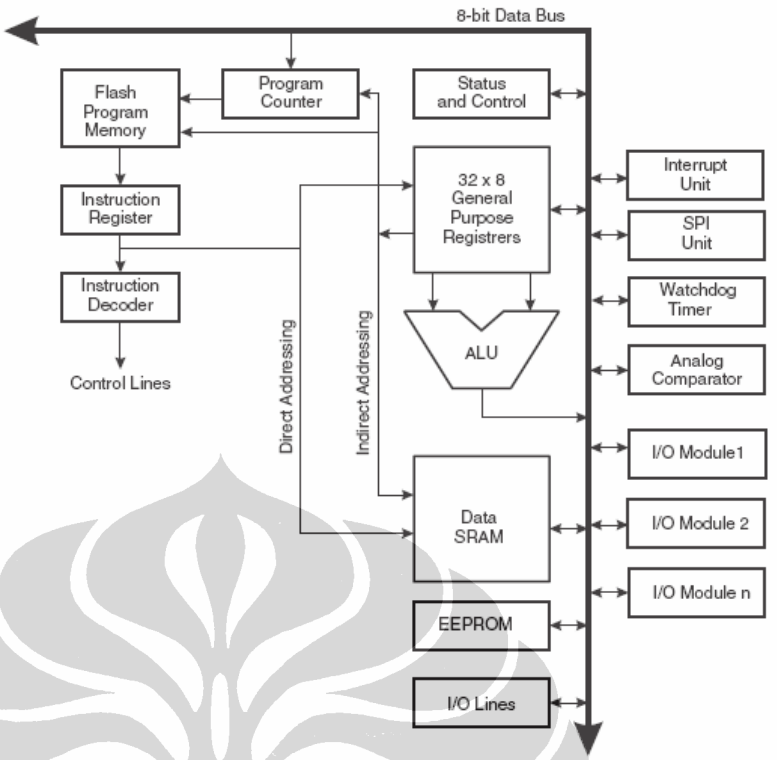
BAB II
PENGENALAN ATMEGA 8535 DAN VHDL

Bab ini akan memaparkan secara rinci mengenai mikrokontroler Atmel ATMega 8535 dan bahasa perangkat keras VHDL yang digunakan dalam tesis ini. Pembahasan mikrokontroler meliputi arsitektur mikrokontroler beserta dengan modul-modul yang dimiliki, jenis pengalamatan yang didukung, dan instruksi-instruksi yang didukung oleh mikrokontroler Atmel ATMega 8535.

2.1 ARSITEKTUR ATMEL ATMEGA 8535

Mikrokontroler ATMega 8535 termasuk dalam keluarga mikrokontroler tipe AVR keluaran Atmel. Istilah AVR sendiri memiliki beberapa pendapat. Meskipun Atmel tidak menjadikan istilah AVR sebagai suatu singkatan, namun beberapa pihak mempunyai pandangan tersendiri. Istilah AVR dapat dihubungkan dengan nama pendesain awal mikrokontroler tersebut yaitu dua orang dari Norwegia bernama Alf Egil Bogen dan Vegard Wollan. Dengan demikian istilah AVR dipanjangkan menjadi "***Alf and Vegard's Risc processor***". Istilah AVR juga dapat diartikan "***Advanced Virtual RISC***" [4].

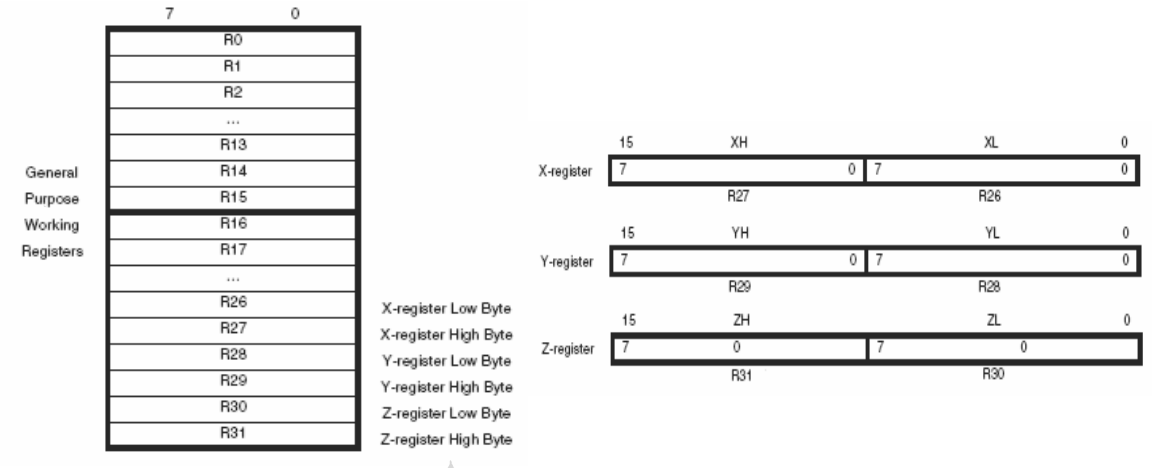
Diagram blok arsitektur dari mikrokontroler Atmel ATMega 8535 secara umum diperlihatkan pada Gambar 2.1. Arsitektur dari mikrokontroler ini secara umum dibuat sedemikian agar mikrokontroler dapat melakukan pengaksesan *memory*, melaksanakan perhitungan, mengontrol modul-modul yang ada, dan menangani *interrupt*.



Gambar 2.1 Diagram blok arsitektur mikrokontroler Atmel ATMega 8535 [5]

Arithmetic Logic Unit (ALU) secara umum menangani operasi aritmatika, logika, dan fungsi-fungsi dalam level *bit*. ALU berhubungan langsung dengan 32x8-bit *General Purpose Registers*. Pada beberapa operasi ALU, dua buah *operand* diambil langsung dari *General Purpose Registers* kemudian hasil operasi disimpan kembali ke *General Purpose Registers*. ALU juga mendukung operasi antara sebuah konstanta dengan sebuah *operand* dari *General Purpose Registers*. Hasil dari operasi aritmatika, logika, dan fungsi-fungsi dalam level *bit* yang dilakukan oleh ALU akan mempengaruhi isi dari *Status Register*.

Gambar 2.2 memperlihatkan susunan dari *General Purpose Registers* yang dimiliki mikrokontroler ATMega 8535. Enam *register* dari 32 *register* dapat dipergunakan sebagai *pointer* untuk melakukan *indirect addressing* ke SRAM. Enam *register* ini terdiri dari tiga pasang *register*, yaitu *register* X, Y, dan Z. *Register* X menempati *register* 26 untuk *low byte* dan *register* 27 untuk *high byte* dan *register* Y menempati *register* 28 untuk *low byte* dan *register* 29 untuk *high byte*. *Register* Z menempati *register* 30 untuk *low byte* dan *register* 31 untuk *high byte*.



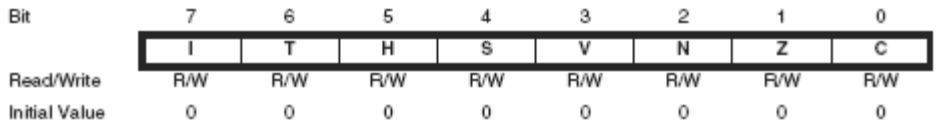
(a) (b)

Gambar 2.2 (a) *General Purpose Registers* (b) X, Y, dan Z *register* [5]

Status Register menyimpan *flag* dari hasil aritmatik, logika, dan operasi *bit* yang dilakukan oleh ALU. *Flag* ini dapat dipergunakan dalam operasi perhitungan tertentu oleh ALU dan dapat dipergunakan untuk mengubah alir program dalam operasi percabangan. Susunan dari *Status Register* dapat dilihat pada Gambar 2.3. *Status Register* menyimpan informasi *flag-flag* sebagai berikut:

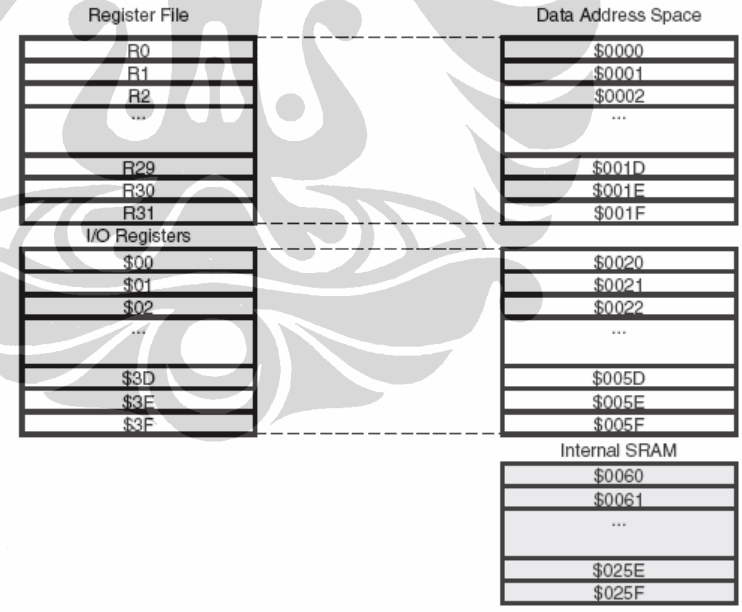
1. I (*Global Interrupt Enable*) digunakan untuk mengaktifkan fungsi *interrupt*. *Flag* ini harus berlogika '1' untuk mengaktifkan fungsi *interrupt* dan sebaliknya. Fungsi *enable* dari masing-masing *interrupt* diatur tersendiri.
2. T (*Bit Copy Storage*). Instruksi BLD (*Bit Load*) dan BST (*Bit Store*) menggunakan *flag* T sebagai sumber dan tujuan dari operasi *bit*. Isi dari *flag* T akan disimpan ke *bit* sebuah *register* dengan instruksi BLD. Sebaliknya instruksi BST akan menyimpan nilai sebuah *bit* dari sebuah *register* ke *flag* T.
3. H (*Half Carry Flag*) digunakan untuk menunjukkan nilai *half carry* dari beberapa operasi aritmatik. Penggunaan *half carry* umumnya pada operasi aritmatik yang menggunakan bilangan *Binary-coded Decimal* (BCD).
4. S (*Sign Bit*) merupakan hasil dari operasi *exclusive or* (Ex-OR) dari *flag* N (*Negative*) dan *flag* V (*Two's Complement Overflow*).
5. N (*Negative*) digunakan untuk menunjukkan hasil bernilai negatif dari hasil operasi aritmatik atau logik.

6. Z (*Zero*) akan bernilai logika ’1’ bila hasil dari operasi aritmatik atau logik bernilai nol.
7. C (*Carry*) menunjukkan adanya *carry* dari hasil operasi aritmatik atau logik.



Gambar 2.3 Susunan *status register* [5]

Mikrokontroler Atmel ATMega 8535 memiliki 608 lokasi *data memory* yang dapat digunakan untuk merujuk *Register File, I/O Memory*, dan internal data SRAM [5]. 96 lokasi awal akan merujuk pada *Register File* dan *I/O Memory* dan 512 lokasi berikutnya akan merujuk pada internal data SRAM. Gambar 2.4 memperlihatkan pengalokasian dari *data memory*.



Gambar 2.4 Alokasi *data memory* [5]

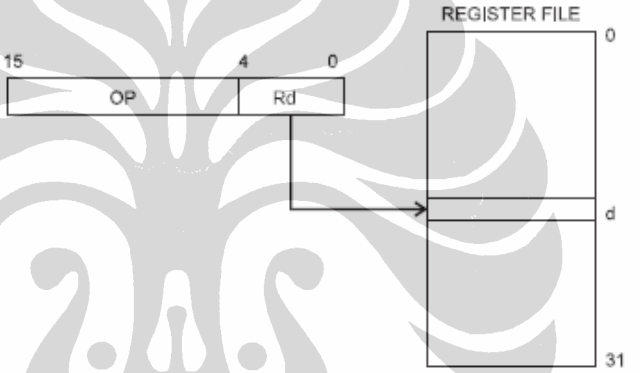
Saat terjadi *interrupt* dan pemanggilan *subroutine*, alamat instruksi selanjutnya dari *Program Counter* akan disimpan sementara di dalam *stack* yang berada di dalam SRAM. Program akan kembali pada alamat yang tersimpan pada

stack setelah rutin *interrupt* atau *subroutine* selesai dilakukan, sehingga program akan dilanjutkan pada baris setelah *interrupt* atau *subroutine* dipanggil.

2.2 JENIS PENGALAMATAN (*ADDRESSING MODE*)

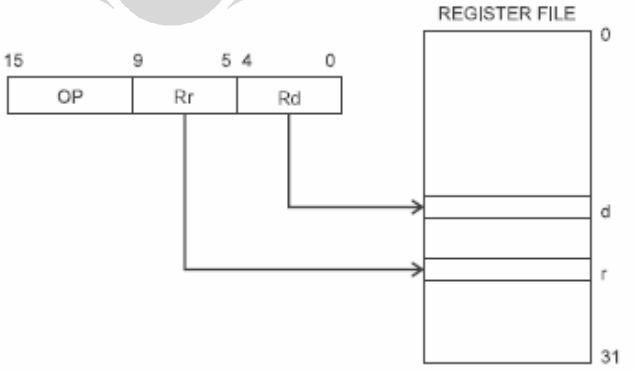
Mikrokontroler Atmel ATMega 8535 mendukung pengalamatan untuk mengakses *program memory* (*Flash*) dan *data memory* (SRAM, *register file*, *I/O memory*). Jenis pengalamatan yang didukung ada 12 buah jenis pengalamatan, yaitu:

1. Pengalamatan *Register Direct, Single Register Rd*. Jenis pengalamatan ini hanya menggunakan sebuah *register*, *Rd*, sebagai sumber dari sebuah *operand*.



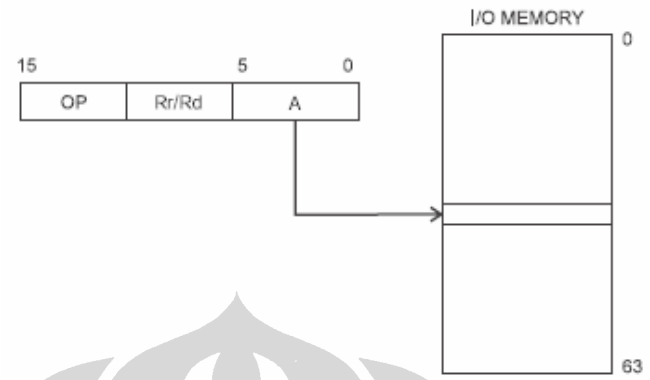
Gambar 2.5 Pengalamatan *register direct, single register Rd* [6]

2. Pengalamatan *Register Direct, Two Register Rd-Rr*. Jenis pengalamatan ini menggunakan dua buah *register*, yaitu *Rd* dan *Rr* sebagai sumber *operand*, yang kemudian hasil operasi akan disimpan kembali pada *register Rd*.



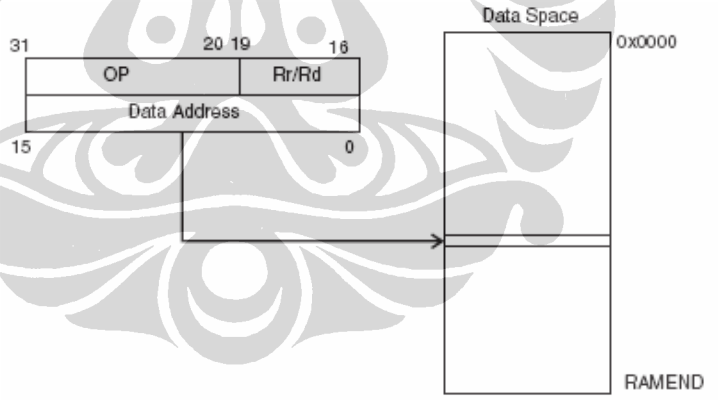
Gambar 2.6 Pengalamatan *register direct, two register Rd-Rr* [6]

3. Pengalamatan *I/O Direct*. Pengalamatan ini menggunakan alamat I/O yang ditunjuk oleh instruksi dan sebuah register sebagai sumber atau tujuan dari hasil operasi.



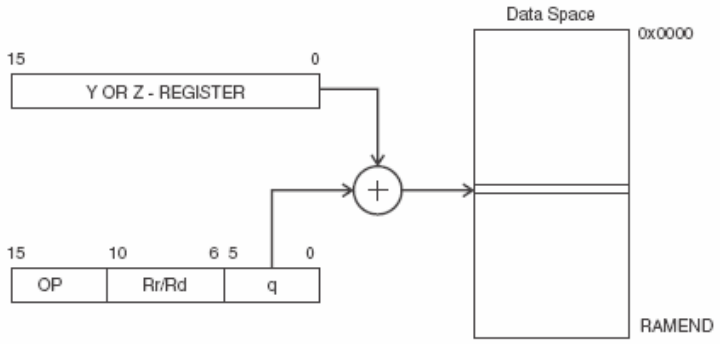
Gambar 2.7 Pengalamatan *I/O direct* [6]

4. Pengalamatan *Data Direct*. Pengalamatan ini menggunakan 32-*bit* pola instruksi dimana alamat dari *data space* berada pada 16-*bit* LSB dari 32-*bit* instruksi tersebut.



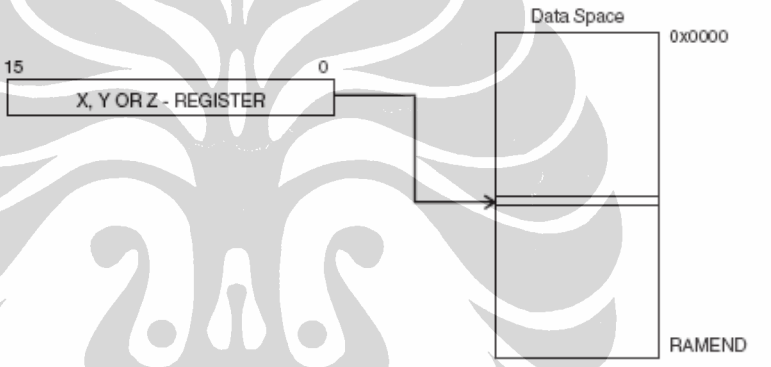
Gambar 2.8 Pengalamatan *relative program* [6]

5. Pengalamatan *Data Indirect with Displacement*. Pengalamatan ini menggunakan isi dari *register* Y atau Z ditambah dengan alamat 6-*bit* dari instruksi untuk menunjukkan alamat dari *data space*.



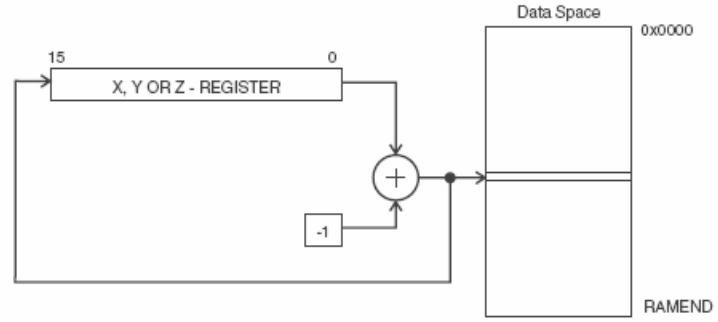
Gambar 2.9 Pengalamatan *data indirect with displacement* [6]

6. Pengalamatan *Data Indirect.* Pengalamatan ini menggunakan isi *register* X, Y, atau Z untuk menunjuk alamat di *data space.*



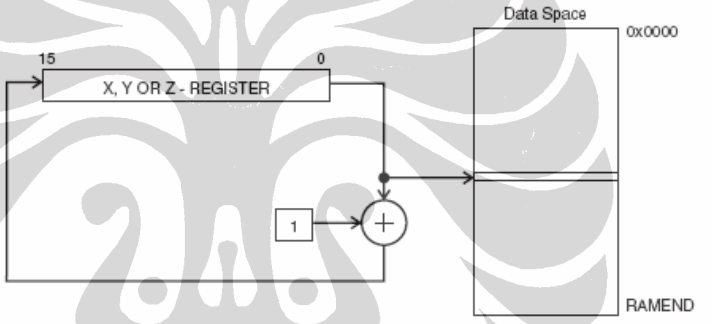
Gambar 2.10 Pengalamatan *data indirect* [6]

7. Pengalamatan *Data Indirect with Pre-decrement.* Pengalamatan ini menggunakan isi dari *register* X, Y, atau Z sebagai alamat dari *data space*. Isi dari *register* X, Y, atau Z akan dikurang 1 dahulu sebelum operasi pengalamatan dilakukan.



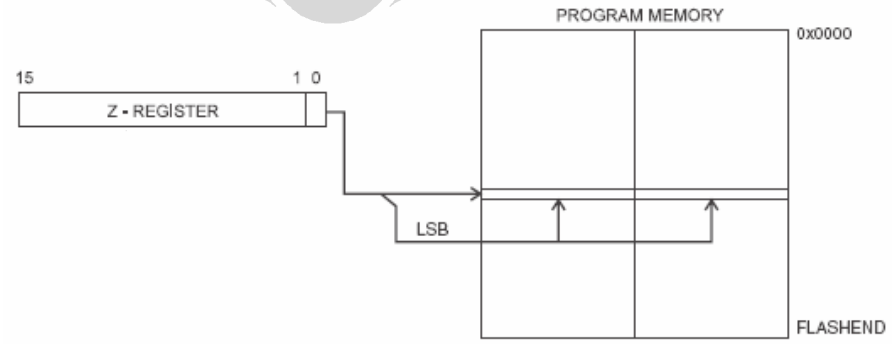
Gambar 2.11 Pengalamatan *data indirect with pre-decrement* [6]

8. Pengalamatan *Data Indirect with Post-increment*. Pengalamatan ini menggunakan isi dari *register* X, Y, atau Z sebagai alamat dari *data space*. Isi dari *register* X, Y, atau Z ditambah dengan 1 setelah operasi pengalamatan dilakukan.



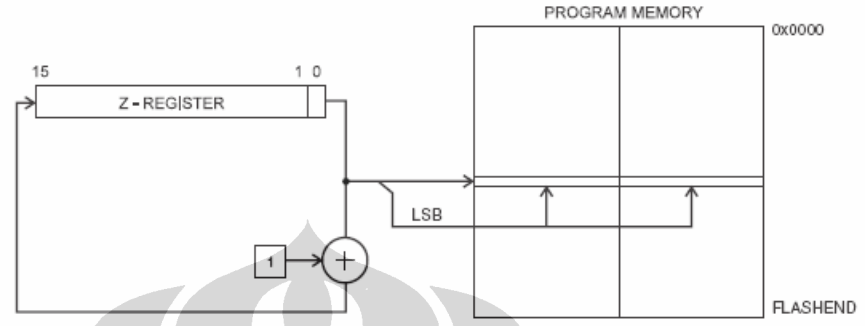
Gambar 2.12 Pengalamatan *data indirect with post-increment* [6]

9. Pengalamatan *Program Memory Constant*. Pengalamatan ini menggunakan isi dari *register* Z untuk menunjuk alamat pada *program memory*.



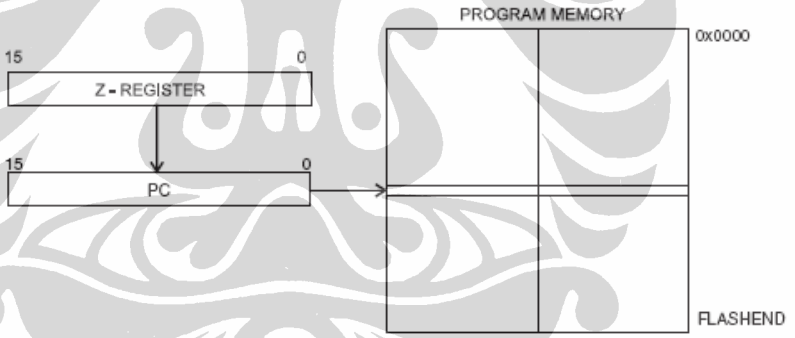
Gambar 2.13 Pengalamatan *program memory constant* [6]

10. Pengalamatan *Program Memory with Post-increment*. Pengalamatan ini menggunakan isi dari *register* Z untuk menunjuk alamat pada *program memory*. Isi dari *register* Z ditambah dengan 1 setelah operasi dilakukan.



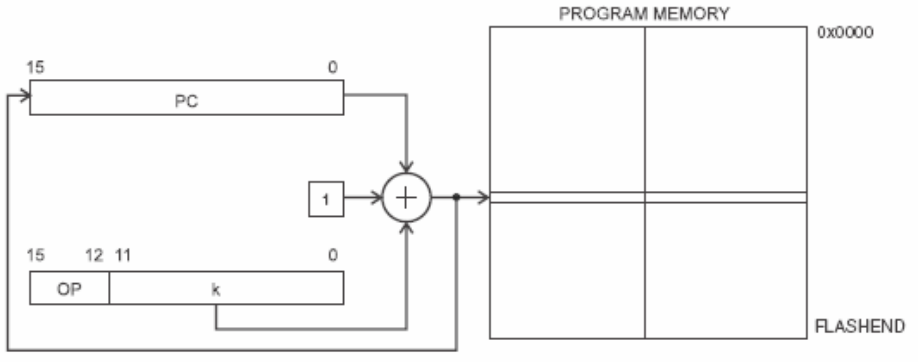
Gambar 2.14 pengalamatan *program memory with post-increment* [6]

11. Pengalamatan *Indirect Program Memory*. Eksekusi program akan dilanjutkan pada alamat yang ditunjuk oleh isi dari *register* Z.



Gambar 2.15 Pengalamatan *indirect program memory* [6]

12. Pengalamatan *Relative Program Memory*. Pengalamatan ini menggunakan langsung isi dari instruksi sebagai alamat pada *program memory*. Eksekusi program akan dilanjutkan pada alamat PC + k + 1 di *program memory*.



Gambar 2.16 Pengalamatan *relative program memory* [6]

2.3 INSTRUKSI ATMEL ATMEGA 8535

Instruksi-instruksi yang dapat ditangani oleh mikrokontroler Atmel ATMega 8535 berjumlah 129 instruksi dan dapat dibagi menjadi empat kategori, yaitu instruksi aritmatik dan logik (***arithmetic and logic instruction***), instruksi percabangan (***branch instruction***), instruksi pindah data (***data transfer instruction***), instruksi bit dan tes bit (***bit and bit-test instruction***), dan instruksi kontrol mikrokontroler (*MCU control instruction*).

Instruksi aritmatik dan logik akan mempengaruhi nilai *flag* yang tersimpan di *Status Register* kecuali untuk instruksi SER (*Set Register*). Kumpulan instruksi ini terdiri dari penjumlahan, pengurangan, perkalian, dan operasi bit. Jumlah instruksi aritmatik dan logik yang dapat ditangani adalah sebanyak 28 buah.

Instruksi percabangan tidak mempengaruhi nilai *flag* yang tersimpan di *Status Register* kecuali untuk instruksi RETI (*Return from Interrupt*), CP (*Compare*), CPC (*Compare with Carry*), dan CPI (*Compare with Immediate*). Instruksi percabangan akan mempengaruhi jalannya program bilamana suatu kondisi terpenuhi. Jumlah instruksi percabangan yang dapat ditangani adalah sebanyak 33 buah yang secara umum dibagi menjadi instruksi percabangan bersyarat (*conditional*) dan tidak bersyarat (*unconditional*).

Instruksi pindah data tidak mempengaruhi nilai *flag* yang tersimpan di *Status Register*. Instruksi-instruksi yang termasuk dalam kelompok ini akan melakukan perpindahan data antar *register*, antara *register* dan SRAM, antara

register dan I/O port. Jumlah instruksi pindah data yang dimiliki adalah sebanyak 35 buah instruksi.

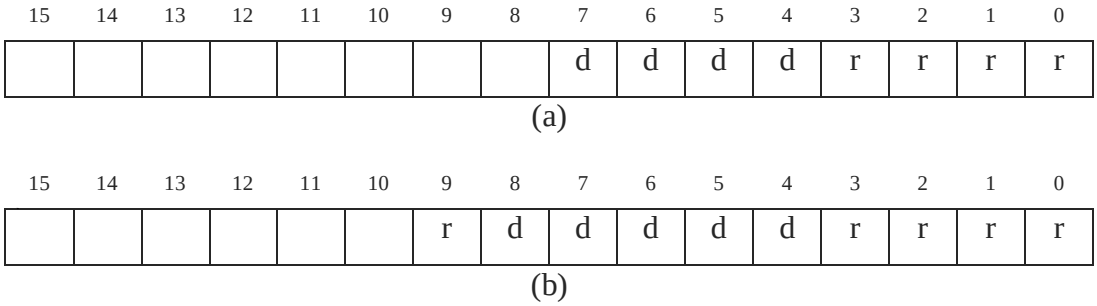
Instruksi bit dan tes bit yang dimiliki oleh ATMega 8535 berjumlah 28 buah instruksi. Kelompok instruksi ini akan mempengaruhi nilai *flag* yang tersimpan di *Status Register*, kecuali instruksi SBI (*Set Bit in I/O Register*), CBI (*Clear Bit in I/O Register*), SWAP (*Swap Nibbles*), dan BLD (*Bit Load from T to Register*).

Instruksi kontrol mikrokontroler berhubungan dengan jalan kerjanya mikrokontroler. Kelompok instruksi yang termasuk instruksi kontrol mikrokontroler tidak mempengaruhi nilai *flag* yang tersimpan di *Status Register*. Jumlah instruksi yang terdapat dalam kelompok ini berjumlah 4 buah.

2.3.1 Pola Instruksi

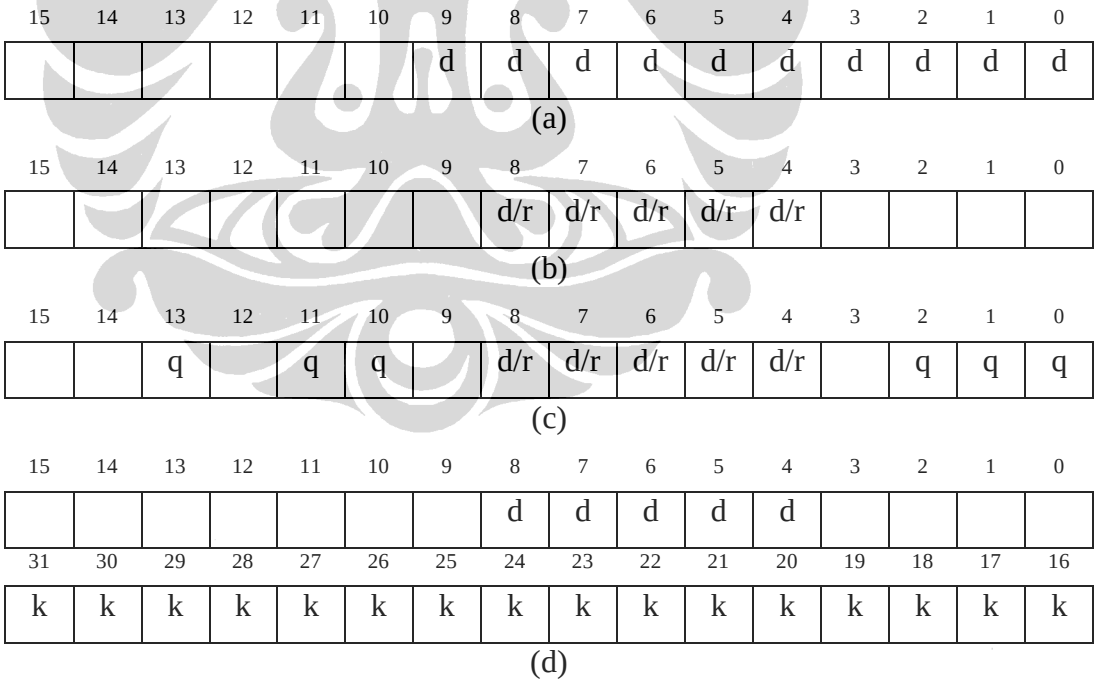
Mikrokontroler Atmel ATMega 8535 memiliki panjang kode instruksi yang tetap, yaitu sebesar 16-bit dan 32-bit dengan pola tertentu. Instruksi-instruksi ini dapat dikelompokkan menjadi 9 kelompok, yaitu pola instruksi yang membutuhkan dua buah register, sebuah register, instruksi untuk operasi yang melibatkan nilai konstanta tertentu, instruksi untuk percabangan tidak bersyarat, instruksi untuk percabangan bersyarat, instruksi untuk operasi yang melibatkan 64 alamat I/O, instruksi untuk operasi yang melibatkan 32 alamat terbawah I/O, instruksi yang berhubungan dengan operasi *bit status register*, dan instruksi yang berkaitan dengan operasi *bit* sebuah *register*.

Pola instruksi yang melibatkan dua buah *register* dapat dilihat pada Gambar 2.17 dimana d adalah *register* tujuan, *r* adalah *register* sumber. Pola instruksi ini dapat dijabarkan menjadi dua buah pola seperti yang ditampilkan pada Gambar 2.17a dan Gambar 2.17b. Instruksi yang mempunyai pola seperti ini adalah MUL, MULS, MULSU, FMUL, FMULS, FMULSU, CPC, SBC, ADD, CPSE, CP, SUB, ADC, AND, EOR, OR, MOVW, dan MOV.



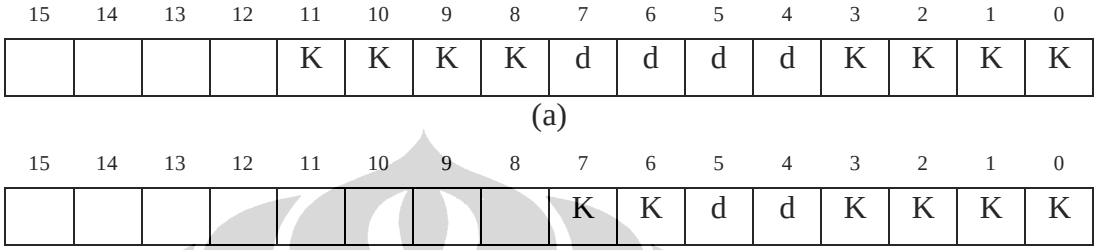
Gambar 2.17 Pola instruksi dengan 2 buah *register* (a) pola 1 (b) pola 2

Pola instruksi yang hanya melibatkan sebuah *register* dapat dilihat pada Gambar 2.18a sampai dengan Gambar 2.18d . Pola instruksi ini juga dapat dibagi dua buah, yaitu instruksi dengan panjang 16-*bit* dan 32-*bit.* Instruksi dengan panjang 16-*bit* seperti instruksi LSL, ROL, TST, CLR, LD, LPM, POP, PUSH, COM, NEG, SWAP, INC, ASR, LSR, ROR, DEC, SER, ,ST, LDD dan STD. Instruksi dengan panjang 32-*bit* mempunyai pola tersendiri seperti pada Gambar 2.18d dan instruksi yang termasuk di dalam pola ini adalah LDS dan STS.



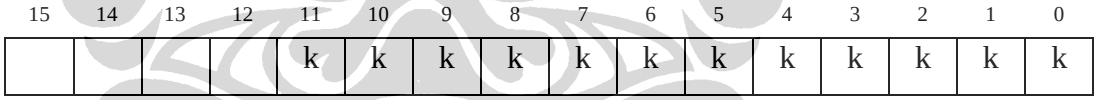
Gambar 2.18 Pola instruksi dengan panjang 16-*bit* (a) pola 1 (b) pola 2 (c) pola 3, (d) pola instruksi dengan panjang 32-*bit*

Pola instruksi untuk operasi yang melibatkan nilai konstanta tertentu melibatkan juga sebuah *register*, pola ini dapat dilihat pada Gambar 2.19 dimana *K* adalah nilai konstanta tertentu dalam *bit* dan *d* adalah *register* tujuan tempat menyimpan hasil operasi. Instruksi yang memiliki pola semacam ini adalah CPI, SBCI, SUBI, ORI, SBR, ANDI, CBR, LDI, ADIW, dan SBIW.



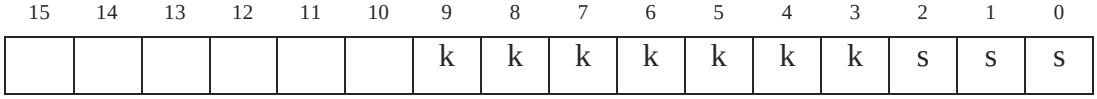
Gambar 2.19 Pola instruksi yang melibatkan nilai konstanta tertentu (a) pola 1 (b) pola 2

Pola instruksi untuk percabangan tidak bersyarat seperti instruksi RJMP dan RCALL dapat dilihat pada Gambar 2.20 dimana *k* adalah nilai alamat tujuan percabangan dalam *bit*.



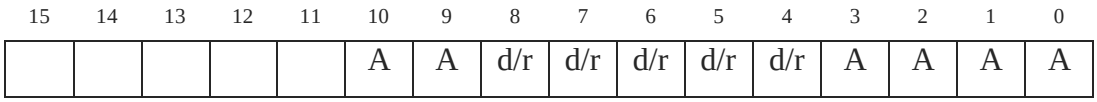
Gambar 2.20 Pola instruksi untuk percabangan tidak bersyarat

Instruksi untuk percabangan bersyarat mempunyai pola seperti pada Gambar 2.21 dimana *k* adalah nilai alamat tujuan percabangan dalam *bit* dan *s* menunjukkan *bit* dalam *status register* yang akan dites. Instruksi yang memiliki pola seperti ini adalah BRCS, BRLO, BREQ, BRMI, BRVS, BRLT, BRHS, BRTS, BRIE, BRBS, BRCC, BRSH, BRNE, BRPL, BRVC, BRGE, BRHC, BRTC, BRID, dan BRBC.



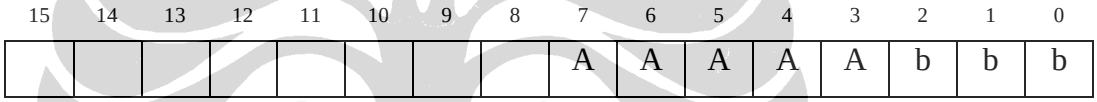
Gambar 2.21 Pola instruksi untuk percabangan bersyarat

Pada instruksi untuk operasi yang melibatkan 64 alamat I/O *register* mempunyai pola seperti pada Gambar 2.22. Contoh dari instruksi ini adalah IN dan OUT yang mengakses *register* I/O. Alamat dari I/O *register* dinyatakan dengan *A-bit*.



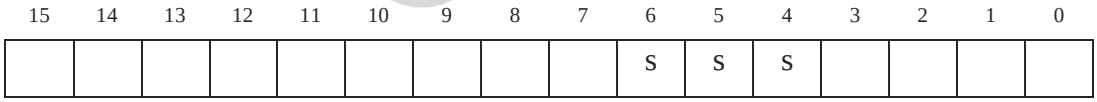
Gambar 2.22 Pola instruksi yang melibatkan 64 alamat I/O *register*

Instruksi untuk operasi yang melibatkan 32 alamat terbawah I/O *register* seperti pada instruksi CBI, SBI, SBIC,dan SBIS mempunyai pola seperti pada Gambar 2.23. *A* adalah alamat I/O *register* dalam bit dan *b* menyatakan *bit* ke-*b* dalam I/O *register* tersebut yang dilakukan operasi.



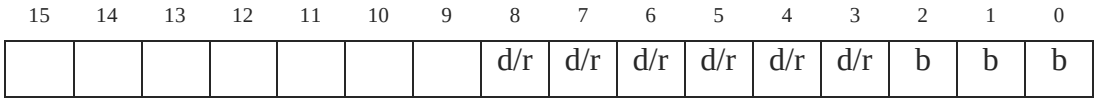
Gambar 2.23 Pola instruksi yang melibatkan 32 alamat terbawah I/O *register*

Untuk instruksi yang berhubungan dengan operasi *bit status register* seperti instruksi BSET, SEC, SEZ, SEN, SEV, SES, SEH, SET, SEI, BCLR, CLC, CLZ , CLN, CLV, CLS, CLH, CLT, dan CLI akan mempunyai pola instruksi seperti pada Gambar 2.24. *s* akan menunjukkan posisi *flag* di dalam *status register* yang akan dilakukan operasi.



Gambar 2.24 Pola instruksi berhubungan dengan operasi *bit status register*

Terakhir instruksi yang berkaitan dengan operasi *bit* pada sebuah *register* pada *General Purpose Registers* seperti instruksi BLD, BST, SBRC, dan SBRS mempunyai pola instruksi seperti pada Gambar 2.25.



Gambar 2.25 Pola instruksi berkaitan dengan operasi *bit* pada sebuah *register*

2.3.2 Pola Instruksi Yang Ekivalen

Dilihat dari 129 instruksi dengan pola yang berbeda-beda, terdapat beberapa instruksi saling berbagi kode mesin yang sama. Hal ini dikarenakan beberapa instruksi melakukan operasi yang sebenarnya sama. Dengan demikian dapat dilakukan pengelompokkan instruksi-instruksi dengan operasi yang sama. Tabel 2.1 menampilkan pengelompokkan instruksi-instruksi ini beserta dengan kode mesinnya.

Tabel 2.1 Pengelompokkan Instruksi Yang Ekivalen

No	Instruksi	Ekivalen	Kode Mesin 16-Bit			
1	ADD	LSL	0000	11rd	dddd	rrrr
2	ADC	ROL	0001	11rd	dddd	rrrr
3	AND	TST	0010	00rd	dddd	rrrr
4	EOR	CLR	0010	01rd	dddd	rrrr
5	ANDI	CBR	0111	KKKK	dddd	KKKK
6	ORI	SBR	0110	KKKK	dddd	KKKK
7	BSET	SEC,SEZ,SEN,SEV, SES,SEH,SET,SEI	1001	0100	0sss	1000
8	BCLR	CLC,CLZ,CLN,CLV, CLS,CLH,CLT,CLI	1001	0100	1sss	1000
9	SER	LDI	1110	KKKK	dddd	KKKK
10	BRCS	BRLO,BREQ,BRMI, BRVS,BRLT,BRHS, BRTS,BRIE,BRBS	1111	00kk	kkkk	ksss
11	BRCC	BRSH,BRNE,BRPL, BRVC,BRGE,BRHC, BRTC,BRID,BRBC	1111	01kk	kkkk	ksss

Berdasarkan Tabel 2.1 maka dari 129 instruksi terdapat 41 buah instruksi yang saling berbagi kode mesin dengan beberapa instruksi lainnya.

2.4 VHSIC HARDWARE DESCRIPTION LANGUAGE (VHDL)

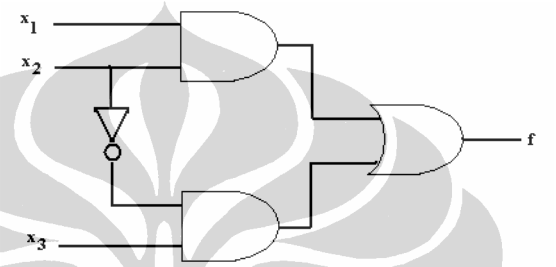
Sebuah sistem dijital dapat dideskripsikan sebagai proses perpindahan *register* dengan menggunakan bahasa deskripsi perangkat keras [7]. *Very High Speed Integrated Circuit* (VHSIC) dikembangkan dalam rangka kebutuhan para perancang rangkaian dijital yang membutuhkan sarana untuk mendeskripsikan struktur dan fungsi dari suatu rangkaian terpadu. Penggunaan VHSIC memungkinkan para perancang untuk melakukan sintesis dan simulasi dari rancangan rangkaian terpadu yang dikembangkan. Dengan demikian maka proses produksi dapat lebih efisien karena hasil rancangan dapat diverifikasi terlebih dahulu sebelum direalisasikan.

VHSIC *Hardware Description Language* (VHDL) yang merupakan bagian dari VHSIC dikembangkan untuk meningkatkan proses pendesainan suatu rangkaian terpadu. Bahasa perangkat keras ini dikembangkan pada awal tahun 1980 di bawah pendanaan dari departemen pertahanan Amerika Serikat. VHDL lahir dari kerjasama para ahli 3 perusahaan besar, yaitu *IBM*, *Texas Instruments*, dan *Intermetrics*.

Versi VHDL yang pertama kali dipublikasikan adalah versi 7.2 yang diterbitkan pada tahun 1985 [8]. Pada tahun 1986, *Institute of Electrical and Electronics Engineers* (IEEE) memaparkan mengenai proposal mengenai standar dari VHDL. Proposal ini selesai pada tahun 1987 setelah mengalami revisi dan diberi nama IEEE 1076-1987. Pada tahun 1994 standar pengganti lahir yaitu IEEE 1076-1993.

VHDL memenuhi beberapa kebutuhan dalam proses pendesainan. Pertama, VHDL dapat mendeskripsikan struktur dari sebuah sistem, yaitu pemecahan menjadi beberapa sub-sistem dan bagaimana sub-sistem tersebut dihubungkan. Kedua, VHDL dapat menspesifikasikan fungsi dari sebuah sistem dalam bentuk bahasa pemrograman yang mudah dimengerti. Ketiga, VHDL memungkinkan simulasi dari desain sebelum dilakukan produksi. Keempat, VHDL memungkinkan struktur desain yang lebih detail untuk disintesis dari sebuah spesifikasi abstrak. Hal ini memungkinkan para perancang untuk berkonsentrasi dalam melakukan pendesainan dan mengurangi waktu dalam proses produksi [9].

VHDL digunakan untuk mendeskripsikan sebuah rangkaian logika. Melalui *compiler* kemudian bahasa VHDL akan diterjemahkan menjadi rangkaian logika. Setiap sinyal logika di dalam rangkaian akan diwakilkan sebagai sebuah objek data oleh VHDL [10]. Contoh sederhana dalam mendeskripsikan sebuah rangkaian dengan menggunakan bahasa VHDL ditampilkan sebagai berikut. Rangkaian logika sederhana yang hendak diterjemahkan dalam bahasa VHDL dan program VHDL yang mendeskripsikannya diperlihatkan pada Gambar 2.26.




```
ENTITY example1 IS
  PORT ( x1, x2, x3 : IN BIT;
         f          : OUT BIT);
END example1;

ARCHITECTURE LogicFunc OF example1 IS
BEGIN
  f<= (x1 AND x2) OR (NOT x2 AND x3);
END LogicFunc;
```

Gambar 2.26 Contoh rangkaian logika sederhana dan kode VHDL

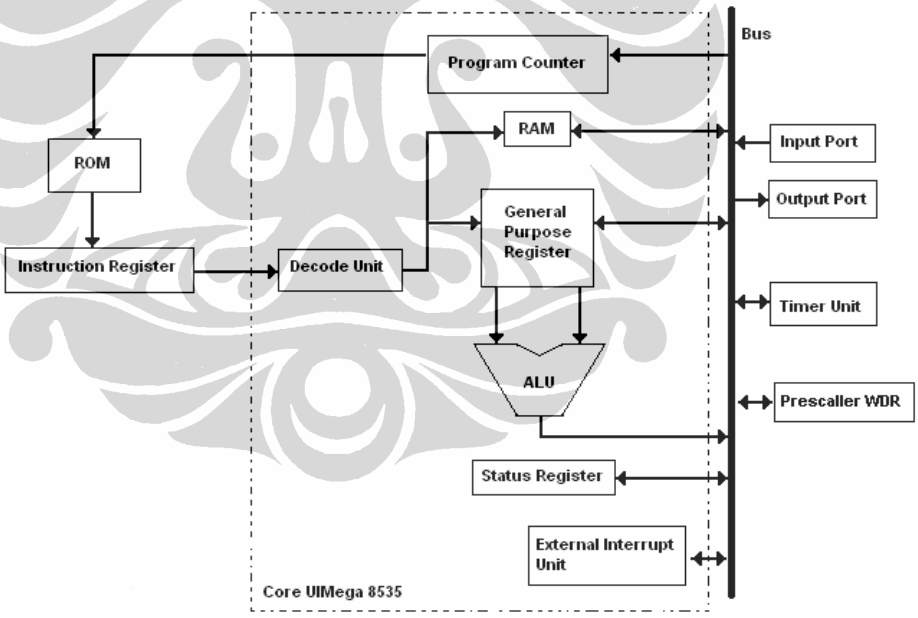
Pada program tersebut, *entity* mendeskripsikan sinyal input dan output untuk rangkaian tersebut. Contoh tersebut memakai example1 sebagai nama *entity*. Sinyal *input* dan *output* untuk *entity* tersebut dinamakan *port* dan diidentifikasikan dengan kata kunci *port*. Setiap *port* harus dispesifikasikan sebagai *input* atau *output* dari *entity* tersebut. Contoh memakai objek *bit* untuk mendefinisikan setiap *port* yang ada. Objek *bit* mempunyai dua nilai, yaitu logika ’1’ dan ’0’.

Karakteristik fungsi dari rangkaian akan didefinisikan di dalam *architecture*. Contoh di atas memakai nama *LogicFunc* untuk *architecture*. Dengan demikian rangkaian di atas mempunyai fungsi *”(x_1 AND x_2) OR (NOT x_2 AND x_3)*” yang kemudian *output* dari fungsi diberikan pada *output f*.

BAB III
PERANCANGAN UIMEGA 8535

3.1 ARSITEKTUR UIMEGA 8535

Arsitektur UIMega 8535 secara umum diperlihatkan pada Gambar 3.1. UIMega 8535 terdiri dari lima modul utama, yaitu modul ROM, modul *instruction register*, modul *core* UIMega 8535, modul *timer interrupt*, dan modul *prescallerWDR*. Modul *core* UIMega 8535 terdiri dari unit *decode*, unit RAM, unit *General Purpose Register*, *Arithmatic and Logic Unit* (ALU), unit *status register*, unit *program counter*, *stack*, dan unit *External Interrupt*. Masing-masing unit dibuat terpisah secara program dan dihubungkan dengan sinyal-sinyal (*wire*). *Bus* pada Gambar 3.1 merupakan kumpulan *wire* yang menghubungkan antar unit dan saling terpisah antara satu dengan yang lain.



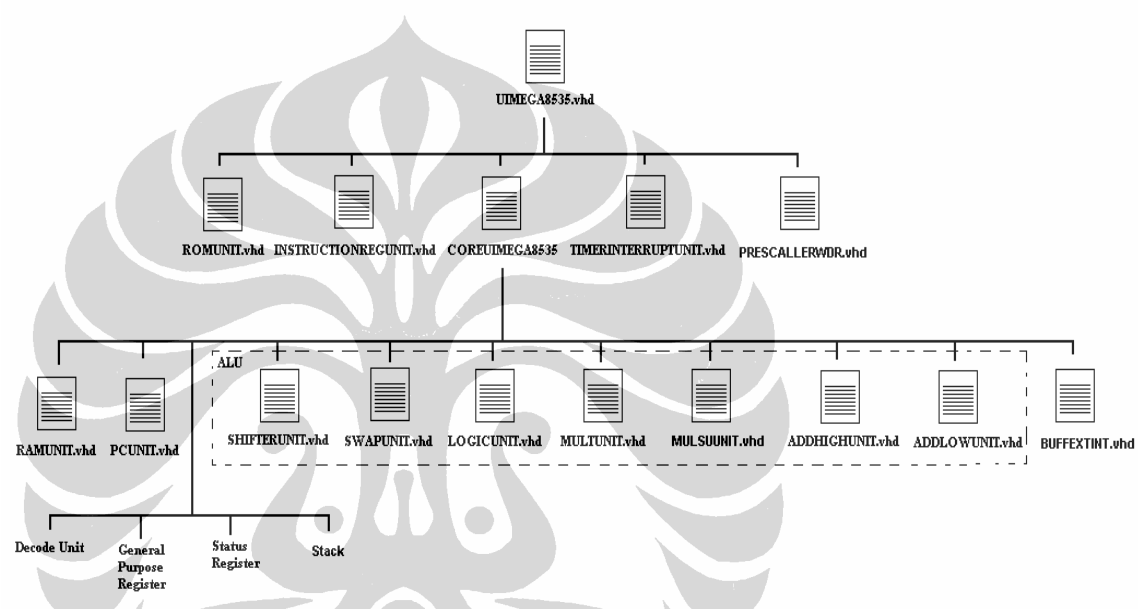
Gambar 3.1 Arsitektur UIMega 8535

3.2 MODUL-MODUL UIMEGA 8535

Hubungan antar modul-modul yang membentuk UIMega 8535 diperlihatkan pada gambar di Lampiran 1.

Hubungan antar *file-file* VHDL diperlihatkan pada Gambar 3.2. Pada tingkat tertinggi adalah *UIMEGA8535.vhd*, dimana *file* ini yang menghubungkan semua *file-file* modul, seperti :

1. *ROMUNIT.vhd*
2. *INSTRUCTIONREGUNIT.vhd*
3. *COREMEGA8535.vhd*
4. *TIMERINTERRUPTUNIT.vhd*
5. *PRESCALLERWDR.vhd*



Gambar 3.2 Hubungan antar *file-file* VHDL

File COREMEGA8535.vhd menghubungkan beberapa *file* unit, seperti :

1. *RAMUNIT.vhd*
2. ALU yang terbentuk dari gabungan *file SHIFTERUNIT.vhd, SWAPUNIT.vhd, LOGICUNIT.vhd, MULTUNIT.vhd, MULSUUNIT.vhd, ADDHIGHUNIT.vhd,* dan *ADDLOWUNIT.vhd*
3. *PCUNIT.vhd*
4. *BUFFEXTINT.vhd*

Beberapa unit terintegrasi langsung di dalam *file COREMEGA8535.vhd*, unit tersebut adalah *General Purpose Register, status register, stack,* dan *decode unit*.

3.2.1 Modul ROM

Modul ROM akan menyimpan program yang akan dijalankan. Program ini disimpan dalam bentuk ekstensi *.hex*. Modul ini mempunyai masukan *pin_pc* yang akan menunjukkan alamat instruksi yang tersimpan di modul ROM dan keluaran *pin_instrom* yang berisi instruksi yang disimpan. Instruksi yang disimpan mempunyai lebar 16-*bit*. Simbol modul ROM diperlihatkan pada Gambar 3.3. Jenis *port*, fungsi, dan konektivitas unit ROM diperlihatkan pada Tabel 3.1. *Source code* modul ini disimpan dalam *file ROMUnit.vhd* pada Lampiran 5.

Gambar 3.3 Modul ROM

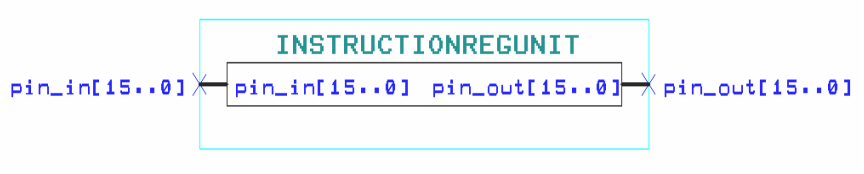
Tabel 3.1 Jenis, Fungsi, dan Konektivitas *Port* Modul ROM

No	Nama *Port*	Jenis *Port*	Fungsi	Terhubung Ke
1	*pin_pc*	masukan	berisi alamat di mana instruksi disimpan di ROM	modul *core* UIMega 8535
2	*pin_datain*	masukan	berisi data yang akan disimpan ke ROM	modul *core* UIMega 8535
3	*pin_werom*	masukan	sinyal penulisan ke ROM	modul *core* UIMega 8535
4	*pin_instrom*	keluaran	berisi instruksi yang dipanggil	modul *instruction register*

3.2.2 Modul *Instruction Register*

Modul *instruction register* berfungsi untuk menyimpan instruksi yang akan dieksekusi oleh modul *core UIMega 8535*. Modul ini juga akan melakukan pembalikan *nibble* dari instruksi. Hal ini ditujukan untuk mempermudah pengkodean dari instruksi oleh *decode unit* di dalam modul *core UIMega 8535*. Acuan pengkodean dapat dilihat pada Lampiran 2. Simbol dari modul *instruction register* diperlihatkan pada Gambar 3.4. Jenis *port*, fungsi, dan konektivitas unit

instruction register diperlihatkan pada Tabel 3.2. *Source code* modul ini disimpan dalam *file InstructionRegUnit.vhd* pada Lampiran 5.



Gambar 3.4 Modul *instruction register*

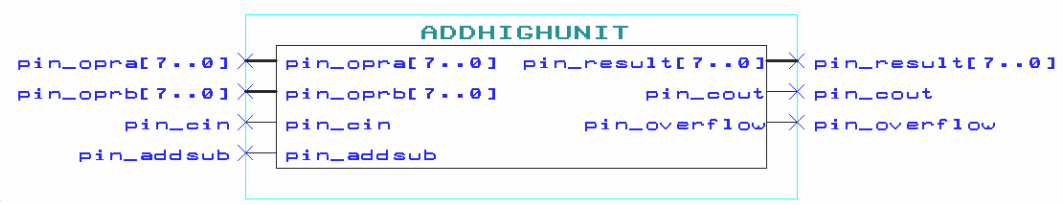
Tabel 3.2 Jenis, Fungsi, dan Konektivitas *Port* Modul *Instruction Register*

No	Nama *Port*	Jenis *Port*	Fungsi	Terhubung Ke
1	*pin_in*	masukan	berisi instruksi yang dipanggil dari ROM	modul ROM
2	*pin_out*	keluaran	berisi instruksi yang dipanggil dan telah dibalik secara *nibble*	modul *core UIMega 8535*

3.2.3 Modul *Core UIMega 8535*

Modul *core UIMega 8535* terdiri dari beberapa unit yaitu unit *decode*, ALU, *General Purpose Register*, RAM, *program counter*, *status register*, *external interrupt*, dan *stack*. Unit *decode* berfungsi untuk melakukan pengkodean terhadap instruksi yang berasal dari ROM untuk kemudian menentukan apa yang selanjutnya harus dilakukan. Unit ini terintegrasi langsung di dalam modul *core* UIMega 8535. Unit ALU terdiri dari beberapa sub-unit, yaitu *adder-subracter*, *multiplier*, *logic*, *swapunit*, dan *shifter*.

Sub-unit *adder-subracter* akan menangani operasi penjumlahan dan pengurangan. Sub-unit ini dibagi menjadi dua buah yaitu sub-unit yang akan menangani pola instruksi 16-*bit* dan 32-*bit* namun masukan dan keluaran dari sub-unit ini sama. Simbol sub-unit ini diperlihatkan pada Gambar 3.5. Jenis *port* dan fungsi sub-unit *adder-subracter* diperlihatkan pada Tabel 3.3. *Source code* modul ini disimpan dalam *file AddLowUnit.vhd* dan *AddHighUnit.vhd* pada Lampiran 5.

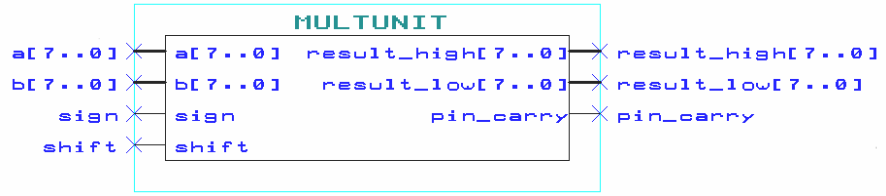


Gambar 3.5 Sub-unit *adder-subracter*

Tabel 3.3 Jenis dan Fungsi *Port* Sub-Unit *Adder-Subracter*

No	Nama *Port*	Jenis *Port*	Fungsi
1	*pin_opra*	masukan	*operand* 1
2	*pin_oprb*	masukan	*operand* 2
3	*pin_cin*	masukan	*carry in*
4	*pin_addsub*	masukan	menentukan operasi penjumlahan (logika ’1’) atau pengurangan (logika ’0’)
5	*pin_result*	keluaran	hasil operasi
6	*pin_cout*	keluaran	*carry out*
7	*pin_overflow*	keluaran	*overflow out*

Sub-unit *multiplier* menangani operasi perkalian. Sub-unit ini terbagi dua, yaitu sub-unit *multunit* yang menangani operasi perkalian bertanda dan sub-unit *mulsuunit* yang menangani operasi perkalian tidak bertanda. Simbol sub-unit *multunit* diperlihatkan pada Gambar 3.6. Jenis *port* dan fungsi sub-unit *multiplier* diperlihatkan pada Tabel 3.4. *Source code* modul ini disimpan dalam *file MultUnit.vhd* pada Lampiran 5.

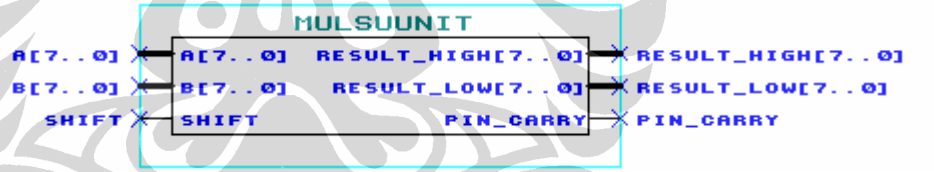


Gambar 3.6 Sub-unit *multunit*

Tabel 3.4 Jenis dan Fungsi *Port* Sub-Unit *Multunit*

No	Nama *Port*	Jenis *Port*	Fungsi
1	*a*	masukan	*operand* 1 (8-*bit*)
2	*b*	masukan	*operand* 2 (8-*bit*)
3	*sign*	masukan	menentukan operasi bertanda (logika ‘1’) atau tidak (logika ‘0’)
4	*shift*	masukan	menentukan operasi membutuhkan pergeseran (logika ’1’) atau tidak (logika ’0’).
5	*result_high*	keluaran	hasil operasi (*bit* ke15 sampai ke 8)
6	*result_low*	keluaran	hasil operasi (*bit* ke 7 sampai ke 0)
7	*pin_carry*	keluaran	*carry out*

Simbol sub-unit *mulsunit* diperlihatkan pada Gambar 3.7. Jenis *port* dan fungsi sub-unit *mulsuunit* diperlihatkan pada Tabel 3.5. *Source code* modul ini disimpan dalam *file MulsuUnit.vhd* pada Lampiran 5.

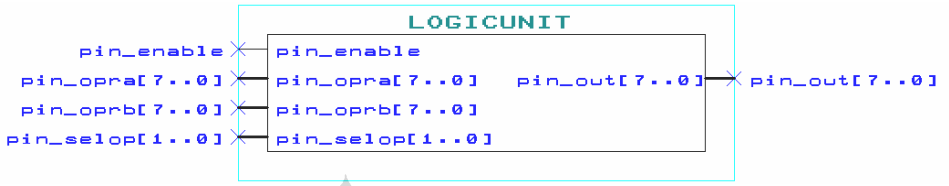


Gambar 3.7 Sub-unit *mulsuunit*

Tabel 3.5 Jenis dan Fungsi *Port* Sub-Unit *Mulsuunit*

No	Nama *Port*	Jenis *Port*	Fungsi
1	*a*	masukan	*operand* 1 (8-*bit*)
2	*b*	masukan	*operand* 2 (8-*bit*)
3	*shift*	masukan	menentukan operasi membutuhkan pergeseran (logika ’1’) atau tidak (logika ’0’).
4	*result_high*	keluaran	hasil operasi (*bit* ke15 sampai ke 8)
5	*result_low*	keluaran	hasil operasi (*bit* ke 7 sampai ke 0)
6	*pin_carry*	keluaran	*carry out*

Sub-unit *logic* akan menangani operasi logika seperti *and, or, ex-or,* dan komplemen. Simbol sub-unit ini diperlihatkan pada Gambar 3.8. Jenis instruksi yang ditangani berdasarkan masukan *pin_selop* diperlihatkan pada Tabel 3.6. Jenis *port* dan fungsi sub-unit *logic* diperlihatkan pada Tabel 3.7. *Source code* modul ini disimpan dalam *file LogicUnit.vhd* pada Lampiran 5.



Gambar 3.8 Sub-unit *logic*

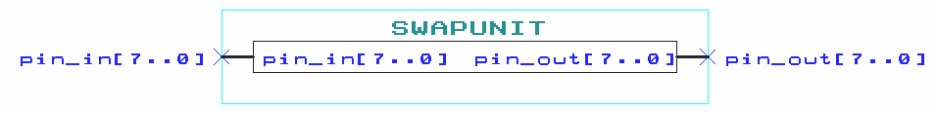
Tabel 3.6 Jenis Operasi Berdasar Masukan *Pin_selop*

Masukan pin_selop	Jenis instruksi
0	AND, ANDI
1	OR, ORI
2	EOR
3	COM

Tabel 3.7 Jenis dan Fungsi *Port* Sub-Unit *Logic*

No	Nama *Port*	Jenis *Port*	Fungsi
1	*pin_enable*	masukan	untuk memfungsikan sub-unit *logic*
2	*pin_opra*	masukan	*operand* 1
3	*pin_oprb*	masukan	*operand* 2
4	*pin_selop*	masukan	untuk menentukan operasi yang dilakukan
5	*pin_out*	keluaran	hasil operasi

Sub-unit *swapunit* khusus menangani instruksi SWAP yang akan membalik *nibble* dari masukan yang diberikan. Simbol sub-unit ini diperlihatkan pada Gambar 3.9. Jenis *port* dan fungsi sub-unit *swapunit* diperlihatkan pada Tabel 3.8. *Source code* modul ini disimpan dalam *file SwapUnit.vhd* pada Lampiran 5.

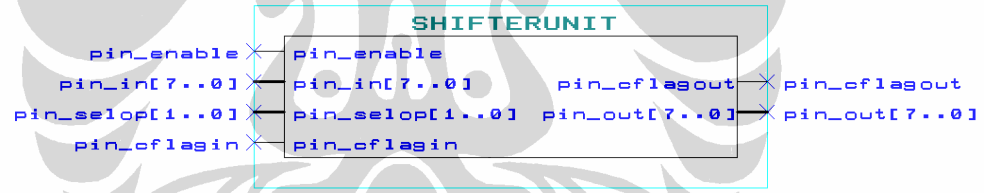


Gambar 3.9 Sub-unit *swapunit*

Tabel 3.8 Jenis dan Fungsi *Port* Sub-Unit *Swapunit*

No	Nama *Port*	Jenis *Port*	Fungsi
1	*pin_in*	masukan	*operand*
2	*pin_out*	keluaran	hasil operasi

Sub-unit ***shifter*** khusus menangani instruksi pergeseran *bit*. Simbol sub-unit ini diperlihatkan pada Gambar 3.10. Jenis instruksi yang ditangani berdasar masukan *pin_selop* diperlihatkan pada Tabel 3.9. Jenis *port* dan fungsi sub-unit *shifter* diperlihatkan pada Tabel 3.10. *Source code* modul ini disimpan dalam *file ShifterUnit.vhd* pada Lampiran 5.



Gambar 3.10 Sub-unit ***shifter***

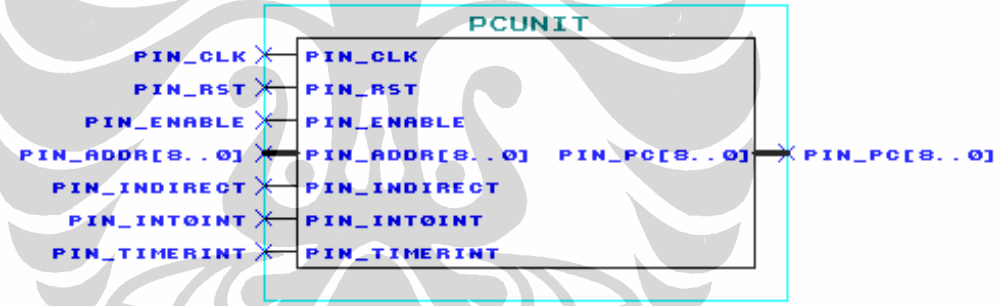
Tabel 3.9 Jenis Operasi Berdasar Masukan *Pin_selop*

Masukan pin_selop	Jenis instruksi
0	LSR
1	ROR
2	ASR

Tabel 3.10 Jenis dan Fungsi *Port* Sub-Unit *Shifter*

No	Nama *Port*	Jenis *Port*	Fungsi
1	*pin_enable*	masukan	untuk memfungsikan sub-unit *shifer*
2	*pin_in*	masukan	*operand*
3	*pin_selop*	masukan	untuk menentukan operasi yang dilakukan
4	*pin_cflagin*	masukan	*flag carry in*
5	*pin_out*	keluaran	hasil operasi
6	*pin_cfagout*		*flag carry out*

Unit *program counter* berfungsi untuk menunjuk alamat instruksi yang tersimpan di unit ROM. Unit ini mampu melakukan pengalamatan sebesar 2^9 *bit* lokasi atau 512 lokasi modul ROM. Simbol unit *program counter* diperlihatkan pada Gambar 3.11. *Source code* modul ini disimpan dalam *file PCUnit.vhd* pada Lampiran 5.



Gambar 3.11 Unit *Program Counter*

Unit ini memiliki masukan *pin_clk, pin_rst, pin_enable, pin_indirect, pin_addr, pin_int0int,* dan *pin_timerint.* Unit ini hanya akan bekerja bila *pin_enable* diberi masukan logika '1'. *Pin_indirect* akan menunjukkan pada unit *program counter* untuk mengambil alamat baru dari pin_*addr*. Masukan *pin_int0int* dan *pin_timerint* akan mengindikasikan adanya *interrupt* sehingga *program counter* perlu melaksanakan rutin *interrupt*. *Pin_int0int* akan mengindikasikan unit *program counter* bahwa terjadi *external interrupt* sedangkan *pin_timerint* mengindikasikan unit *program counter* bahwa terjadi *timer interrupt*. Bilamana terjadi *interrupt* maka alamat eksekusi program yang sedang berlangsung akan disimpan ke dalam *stack* sehingga setelah dilakukan

pemrosesan rutin *interrupt*, eksekusi program dapat dilakukan pada alamat program sebelum dilakukan *interrupt*.

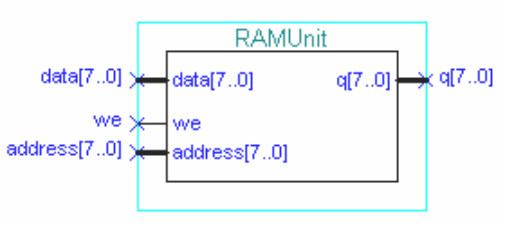
Keluaran *pin_pc* akan menunjukkan alamat instruksi yang akan diambil dari modul ROM. Jenis *port*, fungsi, dan konektivitas unit *program counter* diperlihatkan pada Tabel 3.11.

Tabel 3.11 Jenis, Fungsi, dan Konektivitas *Port Program Counter*

No	Nama *Port*	Jenis *Port*	Fungsi
1	*pin_clk*	masukan	sistem *clock*
2	*pin_rst*	masukan	sistem *reset*
3	*pin_enable*	masukan	untuk memfungsikan unit *program counter*
4	*pin_indirect*	masukan	untuk mengindikasikan *program counter* untuk mengambil alamat dari *pin_addr*
5	*pin_addr*	masukan	alamat percabangan
6	*pin_int0int*	masukan	mengindikasikan terjadi *external interrupt*
7	*pin_timerint*	masukan	mengindikasikan terjadi *timer interrupt*
8	*pin_pc*	keluaran	menunjukkan alamat instruksi di ROM

Unit RAM berfungsi sebagai *data space* atau tempat penyimpanan data bagi mikrokontroler. Unit ini memiliki masukan *data, we,* dan *address.* Keluaran dari unit ini adalah *q*. Masukan dari *address* akan mengalamatkan RAM sehingga isi dari RAM akan langsung dikeluarkan pada keluaran *q.* Proses penulisan dilakukan dengan mengaktifkan masukan *we* dan pemberian data yang yang hendak ditulis pada masukan *data*.

Simbol unit ini diperlihatkan pada Gambar 3.12. Jenis *port* dan fungsi unit RAM diperlihatkan pada Tabel 3.12. *Source code* modul ini disimpan dalam *file RAMUnit.vhd* pada Lampiran 5.



Gambar 3.12 Unit RAM

Tabel 3.12 Jenis, Fungsi, dan Konektivitas *Port* RAM

No	Nama *Port*	Jenis *Port*	Fungsi
1	*address*	masukan	alamat RAM yang hendak dituju
2	*data*	masukan	masukan data yang hendak ditulis pada alamat tertentu dari RAM
3	*we*	masukan	untuk mengaktifkan penulisan pada RAM
4	*q*	keluaran	mengeluarkan isi RAM berdasarkan alamat yang diberikan di masukan *address*

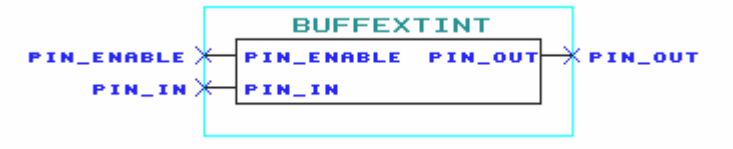
Unit *buffextint* berfungsi untuk mendeteksi *interrupt* dari luar. Unit ini memiliki masukan *pin_enable* dan *pin_in*. Keluaran dari unit ini adalah *pin_out*. Unit ini hanya akan bekerja bila masukan *pin_enable* diaktifkan dari modul *core UIMega 8535*. Masukan *pin_in* dihubungkan ke masukan pin ke-7 dari port A mikrokontroler. Bila unit ini diaktifkan, maka saat terjadi logika *high* pada pin ke-7 dari port A mikrokontroler akan dideteksi sebagai *interrupt*.

Perancangan ini hanya memakai register *General Interrupt Control Register* (GICR) untuk pengaktifan unit ini dengan alamat $3B. Perancangan ini hanya memakai *bit Interrupt Request 0*, yaitu *bit* ke-6 (INT0). Eksternal *interrupt* akan aktif bilamana bernilai logika ’1’ dan nilai *flag* I pada *status register* juga harus berlogika ’1’. Susunan register ini dapat dilihat pada Gambar 3.13.

Bit	7	6	5	4	3	2	1	0	
	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	GICR
Read/Write	R/W	R/W	R/W	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Gambar 3.13 *General Interrupt Control Register* (GICR) [5]

Simbol unit ini diperlihatkan pada Gambar 3.14. Jenis *port* dan fungsi unit *buffextint* diperlihatkan pada Tabel 3.13. *Source code* modul ini disimpan dalam *file buffExtInt.vhd* pada Lampiran 5.

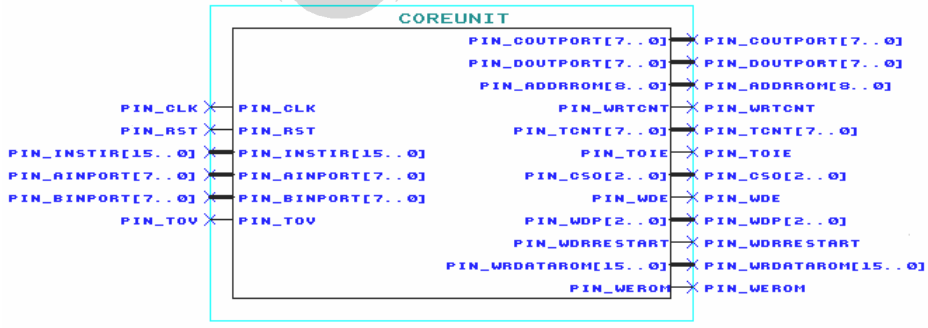


Gambar 3.14 Unit *buffextint*

Tabel 3.13 Jenis, Fungsi, dan Konektivitas *Port Buffextint*

No	Nama *Port*	Jenis *Port*	Fungsi
1	*pin_enable*	masukan	untuk mengaktifkan unit
2	*pin_in*	masukan	untuk mendeteksi kondisi *high* dari pin ke-7 port A
3	*pin_out*	keluaran	untuk mensinyalkan terjadi *external interrupt*

Secara keseluruhan simbol dari modul *core* UIMega 8535 diperlihatkan pada Gambar 3.15. Jenis *port*, fungsi, dan konektivitas modul *core* UIMega 8535 diperlihatkan pada Tabel 3.14. *Source code* modul ini disimpan dalam *file CoreUnit.vhd* pada Lampiran 5.



Gambar 3.15 Modul *core UIMega 8535*

Tabel 3.14 Jenis, Fungsi, dan Konektivitas *Port* Modul *Core* UIMega 8535

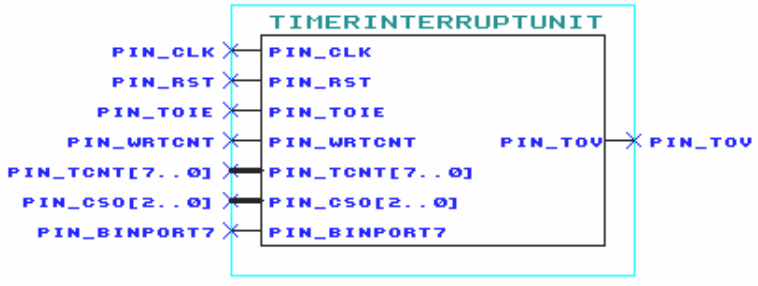
No	Nama *Port*	Jenis *Port*	Fungsi	Terhubung Ke
1	*pin_clk*	masukan	sistem *clock*	sistem *clock* utama
2	*pin_rst*	masukan	sistem *reset*	sistem *reset* utama
3	*pin_instir*	masukan	menerima instruksi dari *instruction register*	modul *instruction register*
4	*pin_Ainport*	masukan	menerima masukan dari *port A*	*port A*
5	*pin_Binport*	masukan	menerima masukan dari *port B*	*port B*
6	*pin_tov*	masukan	menerima sinyal *overflow* dari modul *timer interrupt*	modul *timer interrupt*
7	*pin_Coutport*	keluaran	keluaran ke port C	*port C*
8	*pin_Doutport*	keluaran	keluaran ke port D	*port D*
9	*pin_addrom*	keluaran	keluaran alamat ROM	modul ROM
10	*pin_wrtcnt*	keluaran	sinyal *enable* untuk penulisan nilai awal *timer*	modul *timer interrupt*
11	*pin_tcnt*	keluaran	keluaran nilai awal *timer*	modul *timer interrupt*
12	*pin_toie*	keluaran	sinyal *enable* untuk modul *timer interrupt*	modul *timer interrupt*
13	*pin_cso*	keluaran	keluaran nilai *prescaller* untuk modul *timer interrupt*	modul *timer interrupt*

Tabel 3.14 Jenis, Fungsi, dan Konektivitas *Port* Modul *Core* UIMega 8535 (sambungan)

No	Nama *Port*	Jenis *Port*	Fungsi	Terhubung Ke
14	*pin_wde*	keluaran	sinyal *enable* untuk modul *prescaller watchdog timer*	modul *prescallerWDR*
15	*pin_wdp*	keluaran	keluaran nilai *prescaller* untuk modul *prescallerWDR*	modul *prescallerWDR*
16	*pin_wdrrestart*	keluaran	keluaran sinyal *restart* untuk modul *prescallerWDR*	modul *prescallerWDR*
17	*pin_wrdatarom*	keluaran	berisi data yang akan ditulis ke ROM	modul ROM
18	*pin_werom*	keluaran	sinyal penulisan ke ROM	modul ROM

3.2.4 Modul *Timer Interrupt*

Modul *timer interrupt* berfungsi sebagai *timer* yang dapat digunakan sebagai sumber *interrupt*. Simbol modul diperlihatkan pada Gambar 3.16. Jenis *port*, fungsi, dan konektivitas modul *timer interrupt* diperlihatkan pada Tabel 3.15. *Source code* modul ini disimpan dalam *file TimerInterruptUnit.vhd* pada Lampiran 5.



Gambar 3.16 Modul *timer interrupt*

Tabel 3.15 Jenis, Fungsi, dan Konektivitas *Port* Modul *Timer Interrupt*

No	Nama *Port*	Jenis *Port*	Fungsi	Terhubung Ke
1	*pin_clk*	masukan	sebagai sumber sinyal *clock*	sistem *clock* utama
2	*pin_rst*	masukan	sebagai sumber sinyal *reset*	sistem *reset* utama
3	*pin_toie*	masukan	untuk mengaktifkan unit	modul *core UIMega 8535*
4	*pin_wrtcnt*	masukan	untuk mengaktifkan penulisan nilai awal *timer*	modul *core UIMega 8535*
5	*pin_tcnt*	masukan	masukan nilai awal *timer*	modul *core UIMega 8535*
6	*pin_cso*	masukan	untuk memiilih *clock source*	modul *core UIMega 8535*
7	*pin_Binport7*	masukan	*external source*	*port* B pin 7
8	*pin_tov*	keluaran	sebagai pensinyalan bahwa telah terjadi *overflow*	modul *core UIMega 8535*

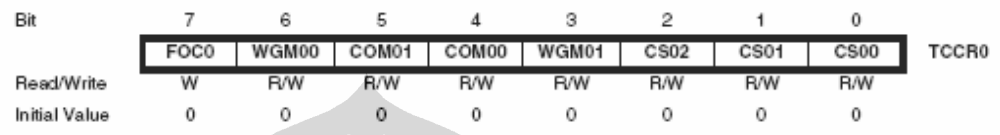
Perancangan ini menggunakan beberapa *register* untuk menjalankan fungsi dari *timer*. *Register-register* tersebut adalah sebagai berikut:

1. *Timer/Counter Interrupt Mask Register* (TIMSK). Register ini memiliki alamat $39. Susunan *bit* pada *register* ini diperlihatkan pada Gambar 3.17. *Bit* ke-0 yaitu *Timer0/Counter0 Compare Match Interrupt Enable* (TOIE0) harus bernilai logika ’1’ dan nilai *flag* I pada *status register* juga harus berlogika ’1’ untuk mengaktifkan *timer interrupt*. *Bit-bit* lain tidak dipakai dalam perancangan ini.

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Gambar 3.17 *Timer/Counter Interrupt Mask Register* (TIMSK) [5]

2. 8-*bit Timer0/Counter0 Register Description* (TCCR0). Register ini memiliki alamat $33. Susunan *bit* pada *register* ini diperlihatkan pada Gambar 3.18. Perancangan ini menggunakan *bit Clock Select*, yaitu *bit* ke-2 (CSO2), ke-1 (CSO1), dan ke-0 (CSO0). Kombinasi dari *bit-bit* ini akan menentukan sumber sinyal *clock* untuk *timer* dan diperlihatkan pada Tabel 3.16.

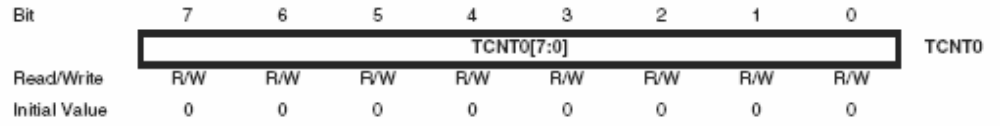


Gambar 3.18 8-*bit Timer/Counter Register Description* [5]

Tabel 3.16 Sumber *Clock Timer* Berdasarkan TCCR0

CSO2	**CSO1**	**CSO0**	**Sumber *clock timer***
0	0	0	*timer* tidak berfungsi
0	0	1	sistem *clock* utama
0	1	0	sistem *clock* utama / 8
0	1	1	sistem *clock* utama / 64
1	0	0	sistem *clock* utama /256
1	0	1	sistem *clock* utama / 1024
1	1	0	*port* eksternal, tepian sinyal turun
1	1	1	*port* eksternal, tepian sinyal naik

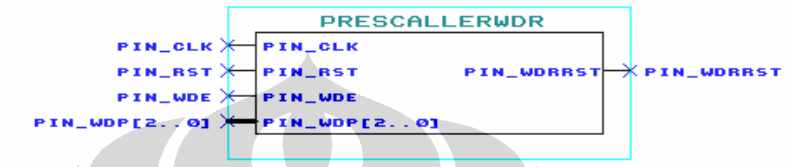
3. *Timer0/Counter0 Register* (TCNT0). Register ini memiliki alamat $32. Isi dari *register* ini dipakai untuk perhitungan *timer* ataupun *counter*. *Register* ini diperlihatkan pada Gambar 3.19.



Gambar 3.19 *Timer0/Counter0 Register* (TCNT0) [5]

3.2.5 Modul *PrescallerWDR*

Modul *prescallerWDR* berfungsi untuk mengatur pewaktuan dari *watchdog timer*. Simbol modul ini dapat dilihat pada Gambar 3.20. Jenis *port*, fungsi, dan konektivitas modul *prescallerWDR* diperlihatkan pada Tabel 3.17. Kombinasi masukan *pin_wdp* menentukan pewaktuan *watchdog timer*. Kombinasi masukan ini dapat dilihat pada Tabel 3.18. *Source code* modul ini disimpan dalam *file prescallerWDR.vhd* pada Lampiran 5.



Gambar 3.20 Modul *prescallerWDR*

Tabel 3.17 Jenis *Port*, Fungsi, Dan Konektivitas Modul *PrescallerWDR*

No	Nama *Port*	Jenis *Port*	Fungsi	Terhubung Ke
1	*pin_clk*	masukan	sebagai sumber *clock*	sistem *clock* utama
2	*pin_rst*	masukan	sebagai sumber *reset*	sistem *reset* utama
3	*pin_wde*	masukan	sistem *enable*	modul *core UIMega 8535*
3	*pin_wdp*	masukan	*prescaller timer*	modul *core UIMega 8535*
4	*pin_wdrrst*	keluaran	sistem reset sistem	semua modul

Tabel 3.18 Sumber *Clock Watchdog Timer* Berdasarkan *Pin_wdp*

Masukan *pin_wdp*	Sumber *Clock*
000	sistem *clock* utama/16
001	sistem *clock* utama/32
010	sistem *clock* utama/64
011	sistem *clock* utama/128
100	sistem *clock* utama/256
101	sistem *clock* utama/512
110	sistem *clock* utama/1024
111	sistem *clock* utama/2048

3.3 REGISTER UIMEGA 8535

UIMega 8535 memiliki beberapa register pendukung untuk kerja dari mikrokontroler. Register-register tersebut adalah *general purpose register* dan I/O register. *General purpose register* UIMega 8535 terdiri dari 16 buah register dan diberi nomor register 0 sampai dengan 15. Register 10 sampai dengan register 15 adalah register khusus yang juga berfungsi sebagai *pointer* dan identik dengan register 26 hingga 31 pada ATMega 8535. Register 10 adalah register X *low byte*, register 11 adalah register X *high byte*, register 12 adalah register Y *low byte*, register 13 adalah register Y *high byte*, register 14 adalah register Z *low byte*, dan register 15 adalah register Z *high byte*. Register X, Y, dan Z digunakan untuk pengalamatan tidak langsung ke RAM.

Pengalamatan ke register hanya membutuhkan 4 *bit* karena hanya berjumlah 16 buah register pada *general purpose register*, oleh karena itu *bit* ke-5 dari register di kode mesin instruksi tidak digunakan.

I/O register dari UIMega 8535 digunakan untuk mengontrol kerja dari beberapa fungsi mikrokontroler, seperti modul *timer interrupt*, external *interrupt*, *watchdog timer*, dan juga untuk penyimpanan *flag* status hasil operasi. Register yang termasuk dalam I/O register dapat dilihat pada Tabel 3.19.

Tabel 3.19 I/O Register UIMega 8535

Nama	Alamat	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SREG	$3F	I	T	H	S	V	N	Z	C
GICR	$3B	-	INT0	-	-	-	-	-	-
TIMSK	$39	-	-	-	-	-	-	-	TOIE
TCCR0	$33	-	-	-	-	-	CS02	CS01	CS00
TCNT0	$32	Timer0 (8 bit)							
WDTCR	$21	-	-	-	-	WDE	WDP2	WDP1	WDP0
PORT A	$1B	A7	A6	A5	A4	A3	A2	A1	A0
PORT B	$18	B7	B6	B5	B4	B3	B2	B1	B0
PORT C	$15	C7	C6	C5	C4	C3	C2	C1	C0
PORT D	$12	D7	D6	D5	D4	D3	D2	D1	D0

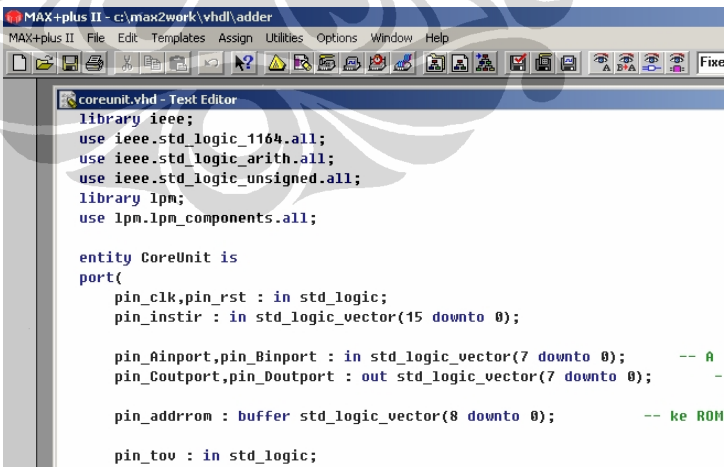
3.4 TAHAPAN PERANCANGAN DAN IMPLEMENTASI DENGAN APLIKASI MAX+PLUS II

Aplikasi MAX+plus II dari Altera memiliki kemampuan untuk proses pendesainan sistem logika yang lengkap, dimulai dari proses pendeskripsian hingga pemrograman *hardware*. Perancangan hingga implementasi dengan menggunakan aplikasi MAX+plus II memiliki beberapa tahapan, yaitu *design entry, project processing, error detection & location, project verification*, dan *device programming*.

3.4.1 *Design Entry*

Pendeskripsian sistem logika yang akan dirancang dilakukan pada tahapan *design entry*. MAX+plus II menyediakan beberapa *editor* untuk mendeskripsikan sistem yang akan dirancang, yaitu *text, waveform, floorplan, graphic* dan *symbol editor*. Pendesain dapat memilih jenis *editor* sesuai dengan keinginan dan kebutuhannya.

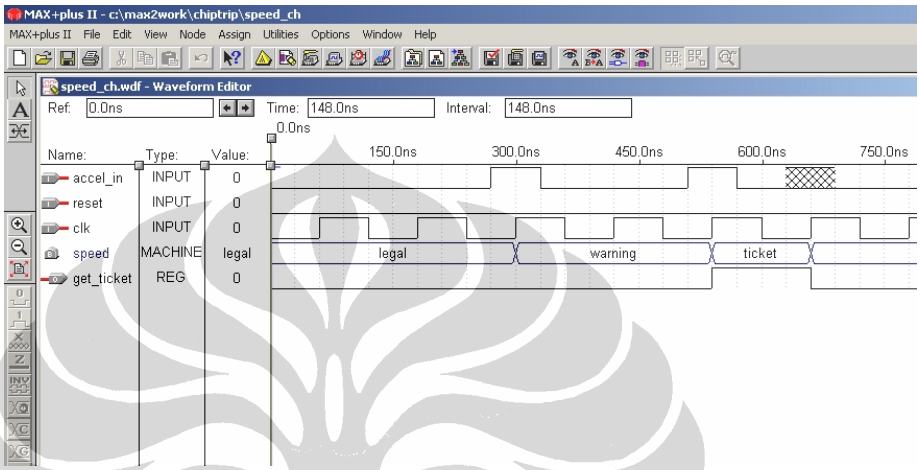
Text editor merupakan *tool* bagi pendesain untuk membuat suatu desain berbasiskan teks/tulisan. MAX+plus II mendukung beberapa bahasa HDL, seperti AHDL, VHDL, dan Verilog HDL. Tampilan *text editor* diperlihatkan pada Gambar 3.21.



Gambar 3.21 Tampilan *text editor*

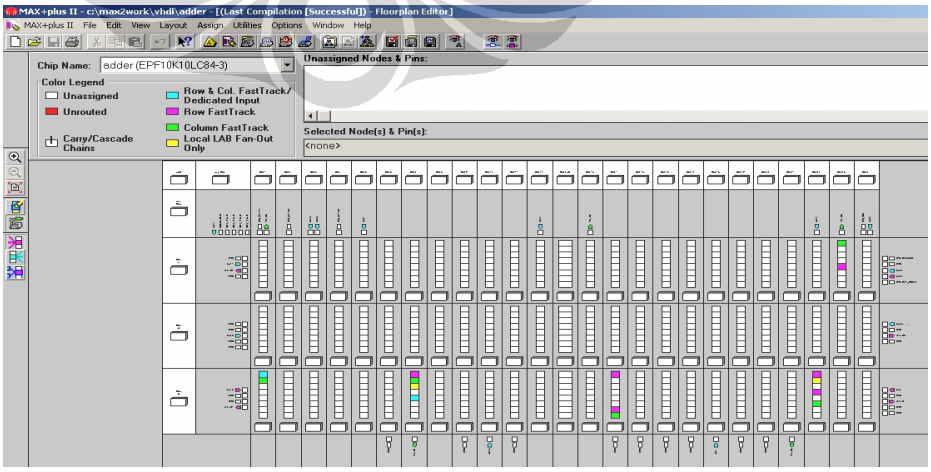
Waveform editor dapat digunakan sekaligus sebagai *editor* maupun sebagai *simulator*. Pendesain dapat menggunakan *waveform editor* untuk

mendefinisikan masukan dalam format logika ataupun *state machine* dan keluaran dalam format *combinatorial, registered,* dan *state machine*. *Waveform editor* dapat melakukan simulasi pewaktuan dengan memberikan masukan yang ada sebuah vektor tes atau kondisi logika tertentu dan diamati keluarannya. Tampilan *waveform editor* dapat dilihat pada Gambar 3.22.



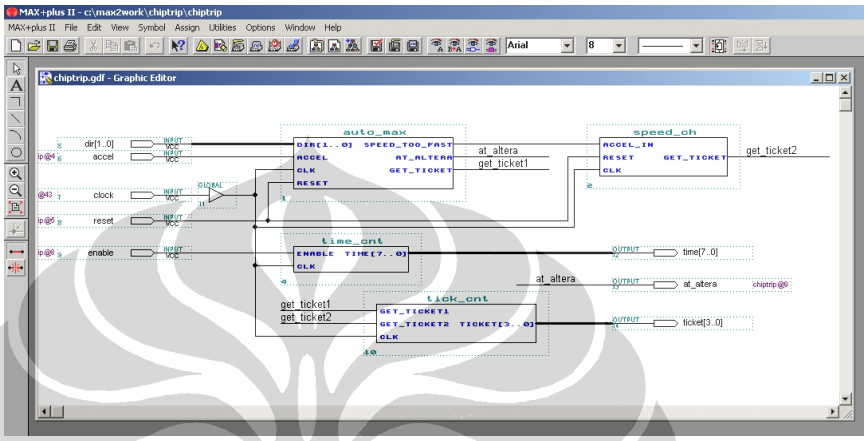
Gambar 3.22 Tampilan *waveform editor*

Floorplan editor menampilkan penggunaan *resources* dari divais target secara fisik. *Floorplan editor* juga menampilkan hasil dari pengalokasian beberapa divais target oleh *compiler*. Tampilan *floorplan editor* diperlihatkan pada Gambar 3.23.



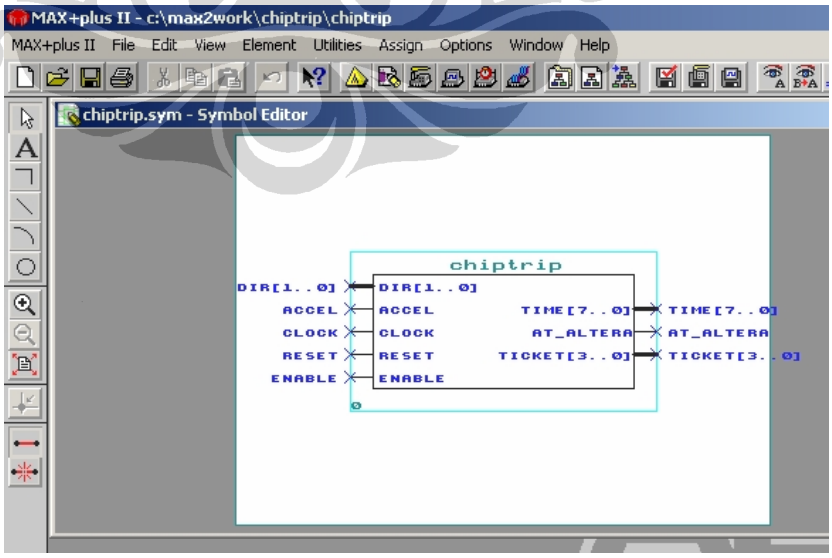
Gambar 3.23 Tampilan *floorplan editor*

Graphic editor memungkinkan pendesain untuk membuat suatu desain yang sederhana maupun kompleks secara cepat. *Graphic editor* dapat menggunakan simbol-simbol bawaan dari MAX+plus II maupun dari simbol yang dibuat oleh pendesain. Tampilan dari *graphic editor* ini dapat dilihat pada Gambar 3.24.



Gambaran 3.24 Tampilan *graphic editor*

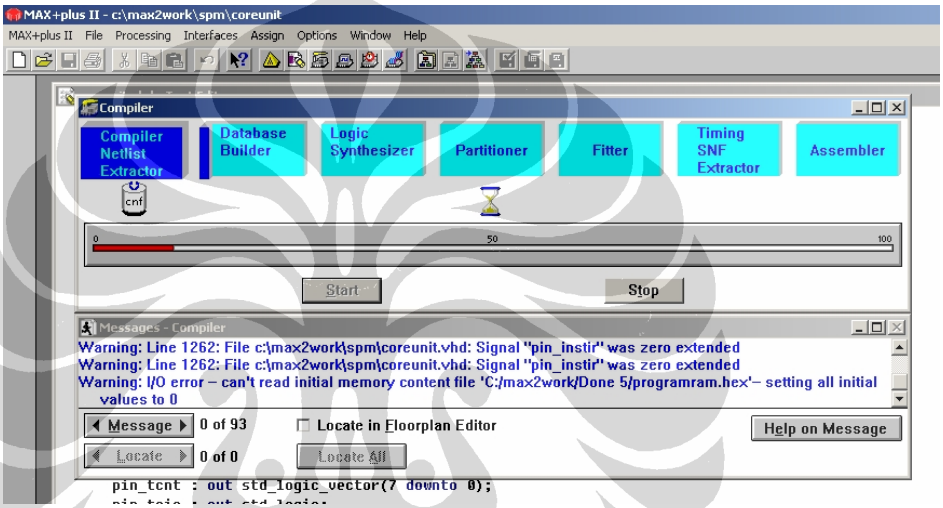
Symbol editor memiliki kemampuan untuk mengedit simbol-simbol yang ada maupun simbol yang dibuat oleh pendesain. Tampilan dari *symbol editor* diperlihatkan pada Gambar 3.25.



Gambar 3.25 Tampilan *symbol editor*

3.4.2 *Project Processing*

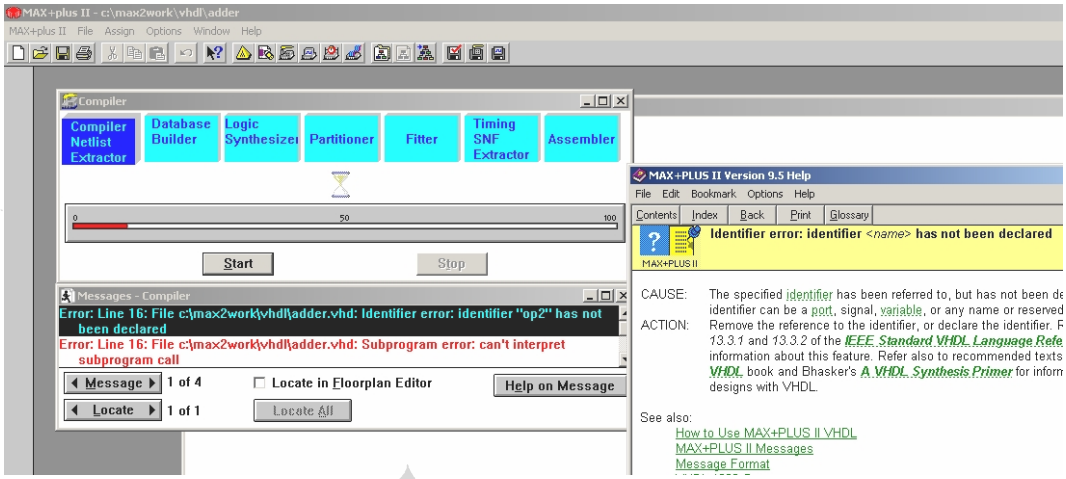
Tahapan *project processing* dilakukan oleh *compiler* aplikasi MAX+plus II. *Compiler* aplikasi MAX+plus II terdiri dari serangkaian modul yang akan melakukan pengecekkan terhadap *error* yang mungkin terjadi, sintesis rangkaian logika, penempatan hasil kompilasi ke sebuah atau beberapa divais Altera, dan sekaligus menghasilkan *file* yang nantinya akan dipergunakan untuk simulasi, analisa waktu, dan pemrograman divais. Gambar 3.26 memperlihatkan proses kompilasi sedang dilakukan.



Gambar 3.26 Tampilan proses kompilasi

3.4.3 *Error Detection & Location*

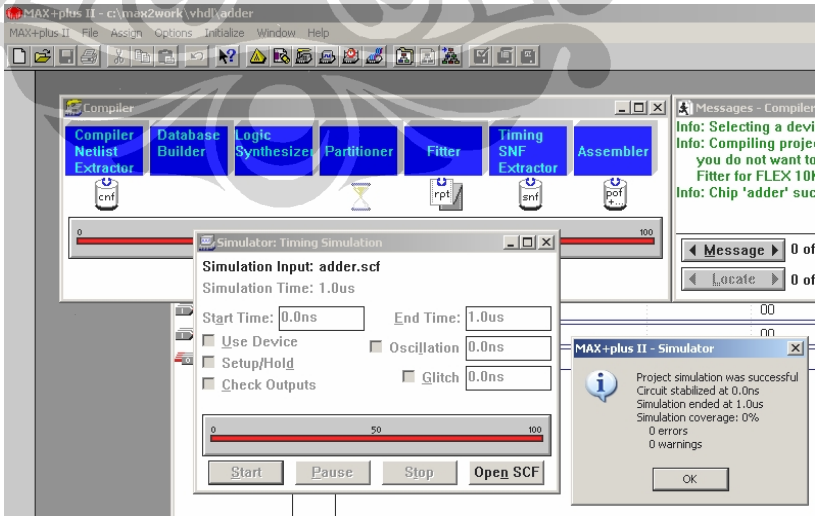
Tahap *error detection & location* merupakan tahap selanjutnya setelah kompilasi dilakukan. Pada tahapan ini dapat diketahui apakah rancangan memiliki *error* atau tidak, bila rancangan memiliki *error* maka aplikasi MAX+plus II dapat menunjukkan di mana *error* itu terjadi pada bagian *message processor*. Bagian *message processor* juga dapat memberikan saran untuk mengatasi *error* yang terjadi. Gambar 3.27 memperlihatkan tampilan bilamana terjadi *error* dan saran dari *message processor* untuk mengatasi *error* tersebut.



Gambar 3.27 Tampilan bila terjadi *error*

3.4.4 *Project Verification*

Tahapan *project verification* dilakukan untuk mengecek apakah desain yang telah dihasilkan sesuai dengan apa yang diinginkan. Tahapan ini menggunakan MAX+plus II Simulator dan *waveform editor* untuk mensimulasikan hasil desain. Verifikasi dilakukan dengan memberikan vektor tes atau kondisi logika tertentu kepada masukan dan diamati keluaran dari desain. Tampilan MAX+plus II Simulator dapat dilihat pada Gambar 3.28.

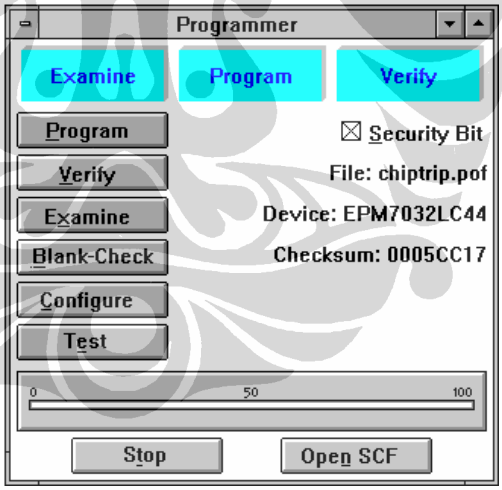


Gambar 3.28 Tampilan MAX+plus II Simulator

3.4.5 *Device Programming*

Tahapan ini akan melakukan *download* data konfigurasi dari hasil rancangan untuk memprogram divais target. Tahapan ini membutuhkan kabel data khusus dari Altera yang dihubungkan ke *development board* tempat divais target berada. Kabel-kabel data ini yaitu diantaranya adalah *BitBlaster, ByteBlaster,* dan *ByteBlasterMV*. Perbedaan jenis kabel *download* ini terletak pada jenis kabel dan tipe divais target yang didukung.

Kabel *BitBlaster* digunakan untuk memprogram atau mengkonfigurasi divais target Altera tipe MAX9000, MAX7000S, MAX7000A, FLEX10K, FLEX8000, dan FLEX6000, melalui *serial port* RS-232 standar. Kabel *ByteBlaster* mendukung semua divais target yang didukung oleh kabel *BitBlaster,* ditambah divais target MAX9000A. Kabel *ByteBlasterMV* mendukung semua divais target yang didukung oleh kabel *ByteBlaster*, ditambah divais target FLEX10KV, FLEX10KA, dan FLEX10KB. Tampilan MAX+plus II *Hardware Programmer* diperlihatkan pada Gambar 3.29.



Gambar 3.29 Tampilan MAX+plus II *Hardware Programmer*

Tesis ini hanya mencapai tahapan *project verification* dan tidak sampai tahapan *device programming*. Simulasi dan verifikasi dilakukan hingga pada simulasi pewaktuan yang terdapat pada *waveform editor*.

BAB IV
HASIL VERIFIKASI UIMEGA 8535

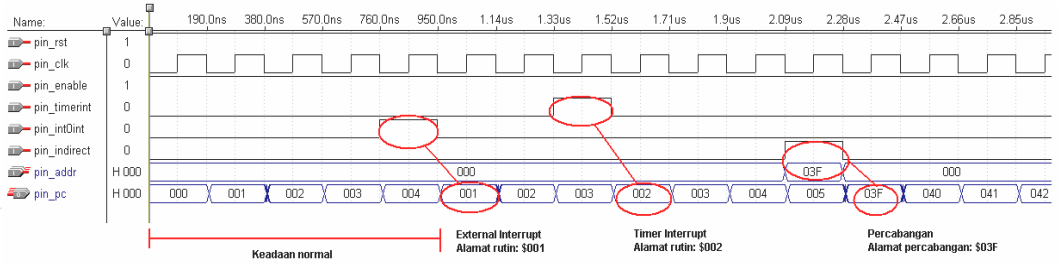
Bab ini memaparkan hasil verifikasi perancangan yang dilakukan. Verifikasi dilakukan dengan menggunakan simulasi melalui diagram pewaktuan dari aplikasi MAX+plus II. Metode yang dipakai adalah dengan melakukan verifikasi terhadap masing-masing modul yang membentuk UIMega 8535. Verifikasi dilakukan kembali setelah penggabungan semua modul sehingga membentuk UIMega 8535 dengan memberikan semua instruksi yang dimiliki oleh mikrokontroler Atmel ATMega 8535.

4.1 VERIFIKASI MODUL-MODUL UIMEGA 8535

4.1.1 Modul *Core UIMega 8535*

4.1.1.1 Unit Program Counter

Verifikasi terhadap unit *program counter* dilakukan dengan melakukan beberapa kemungkinan. Kemungkinan pertama adalah unit *program counter* bekerja dalam keadaan normal, yaitu tidak terjadi *interrupt* atau kondisi percabangan sehingga keluaran *pin_pc* akan terus bertambah satu secara bertahap yang menunjukkan alamat instruksi di ROM. Kemungkinan kedua adalah terjadi *interrupt*, yaitu *external interrupt* dan *timer interrupt. External interrupt* akan menyebabkan unit *program counter* memanggil alamat instruksi di 0x001, sedangkan *timer interrupt* akan memanggil alamat instruksi di 0x002. Kemungkinan terakhir adalah terjadi percabangan, sehingga *program counter* akan memanggil alamat instruksi berdasarkan alamat percabangan yang diberikan. Hasil simulasi unit ini diperlihatkan pada Gambar 4.1.



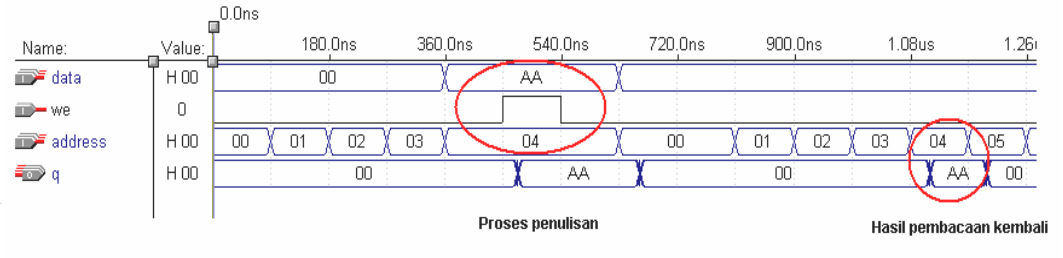
Gambar 4.1 Hasil simulasi unit *program counter*

Waktu awal simulasi *program counter* diberi diinisialisasi dengan memberikan masukan *master_rst* logika ’0’ (aktif *low*). Setelah diinisialisasi maka keluaran dari *program counter* akan bertambah satu setiap tepian naik sinyal *clock*. Pada saat terjadi *external interrupt*, maka masukan *pin_int0int* bernilai logika ’1’, sehingga *program counter* akan menunjuk alamat rutin *interrupt* yaitu 0x001. Selanjutnya *program counter* akan kembali bertambah satu setiap tepian turun sinyal *clock*. Bila terjadi *timer interrupt*, maka masukan *pin_timerint* bernilai logika ’1’, sehingga *program counter* akan menunjuk alamat rutin *interrupt* yaitu 0x002. Selanjutnya *program counter* akan kembali bertambah satu setiap tepian naik sinyal *clock*.

Bilamana terdapat instruksi percabangan, maka masukan *pin_rjmprcallretreti* bernilai logika ’1’, sehingga menyebabkan *program counter* menunjuk pada alamat percabangan. Pada Gambar 4.1 disimulasikan terjadi percabangan dengan alamat percabangan adalah 0x03F.

4.1.1.2 Unit RAM

Verifikasi terhadap unit RAM dilakukan dengan membaca isi RAM kemudian dilakukan proses penulisan data ke alamat tertentu di RAM. Proses pembacaan dilakukan kembali pada alamat yang telah ditulis data. Hasil verifikasi dan simulasi unit RAM dapat dilihat pada Gambar 4.2.



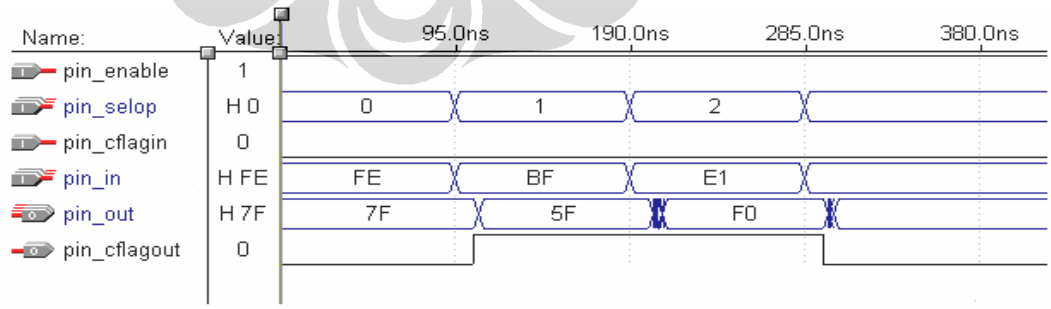
Gambar 4.2 Hasil simulasi unit RAM

Unit RAM secara langsung akan mengeluarkan isi dari alamat yang diberikan. Pada awal simulasi RAM masih kosong. Proses penulisan berlangsung ketika masukan *we* diberi logika *high* sehingga RAM langsung menulis data yang tersedia di masukan *data* dengan alamat yang terdapat pada masukan ***address***. Simulasi menunjukkan proses penulisan dilakukan pada alamat $04 dengan data $AA.

Proses pembacaan dilakukan kembali pada alamat $04 dan pada alamat tersebut telah terisi data $AA yang ditulis tadi.

4.1.1.3 Unit Shifter

Unit *Shifter* digunakan untuk melakukan operasi pergeseran *bit*. Instruksi yang berhubungan dengan unit ini adalah ASR, LSR, dan ROR. Jenis operasi pergeseran berdasarkan masukan pada *pin_selop*. Hasil simulasi dapat dilihat pada Gambar 4.3.



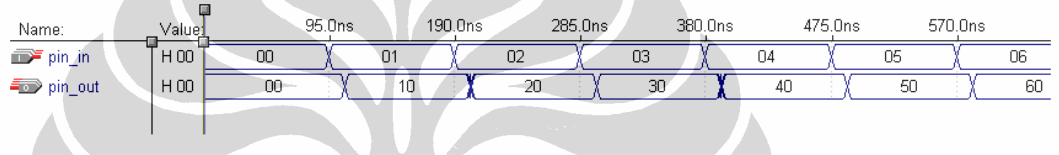
Gambar 4.3 Hasil simulasi unit *shifter*

Saat *pin_selop* diberi masukan 0 maka unit melakukan pergeseran *bit* ke kanan dengan *bit* ke-7 menjadi ’0’ dan *bit* ke-0 dikeluarkan ke *pin_cflagout*

sehingga masukan $FE akan menghasilkan keluaran $7F. Saat *pin_selop* diberi masukan 1 maka unit melakukan pergeseran *bit* ke kanan dengan *bit* ke-7 dari masukan *pin_cflagin* dan *bit* ke-0 dikeluarkan ke *pin_cflagout* sehingga masukan $BF akan menghasilkan keluaran $5F. Saat *pin_selop* diberi masukan 2 maka unit melakukan pergeseran *bit* ke kanan dengan *bit* ke-7 tetap dan *bit* ke-0 dikeluarkan ke *pin_cflagout* sehingga masukan $E1 akan menghasilkan keluaran $F0.

4.1.1.4 Unit Swap

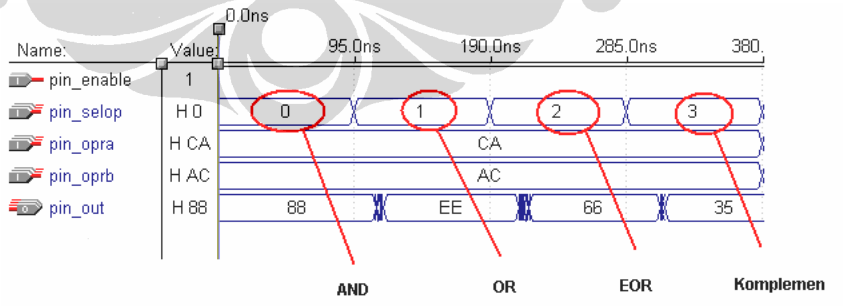
Unit *swap* digunakan untuk melakukan pertukaran *nibble* masukan dari *pin_in*. Hasil simulasi unit ini diperlihatkan pada Gambar 4.4.



Gambar 4.4 Hasil simulasi unit *swap*

4.1.1.5 Unit Logic

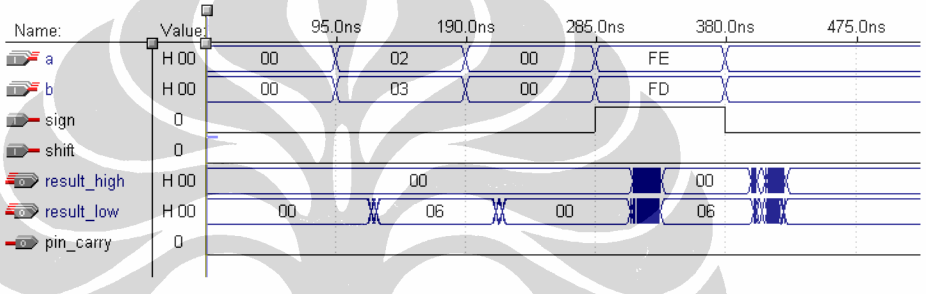
Unit *logic* digunakan untuk melakukan operasi logika, seperti *AND, OR, Exclusive-OR,* dan komplemen. Jenis operasi berdasarkan masukan pada *pin_selop*. Hasil simulasi diperlihatkan pada Gambar 4.5.



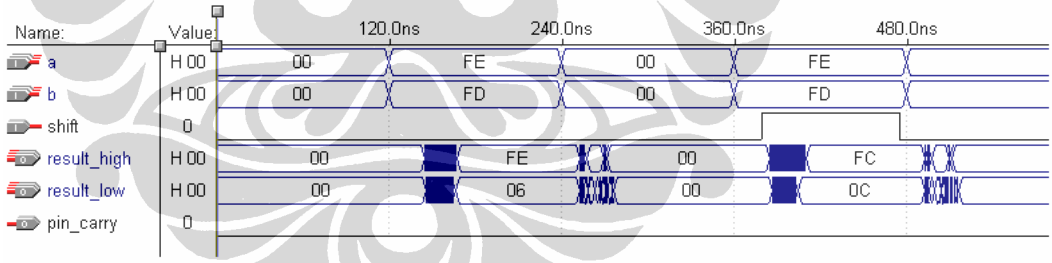
Gambar 4.5 Hasil simulasi unit *logic*

4.1.1.6 Unit Multiplier

Unit *multiplier* dapat dibedakan menjadi dua buah, yaitu *multunit*, yang akan menangani instruksi MUL, MULS, FMUL, dan FMULSU, dan *mulsuunit*, yang akan menangani instruksi MULSU dan FMULSU. Operasi *multunit* memerlukan dua buah operand tidak bertanda atau keduanya bertanda, sedangkan operasi yang ditangani *mulsuunit* memerlukan dua buah operand yang salah satunya bertanda dan lainnya tidak bertanda. Hasil simulasi *multunit* diperlihatkan pada Gambar 4.6. Hasil simulasi *mulsuunit* diperlihatkan pada Gambar 4.7.



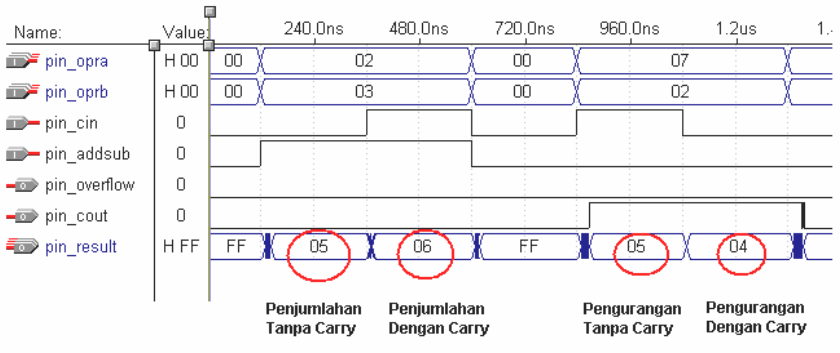
Gambar 4.6 Hasil simulasi unit *multunit*



Gambar 4.7 Hasil simulasi unit *mulsuunit*

4.1.1.7 Unit Adder-Subracter

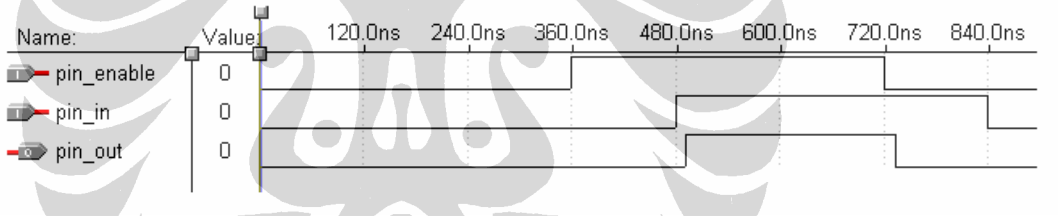
Unit *adder-subracter* menangani operasi penambahan dan pengurangan. Unit ini dibedakan menjadi dua, yaitu untuk menangani operasi instruksi 16-*bit* dan untuk menangani operasi instruksi 32-*bit*, namun susunan masukan dan keluaran sama. Hasil simulasi unit ini dapat dilihat pada Gambar 4.8.



Gambar 4.8 Hasil simulasi unit *adder-subracter*

4.1.1.8 Unit External Interrupt

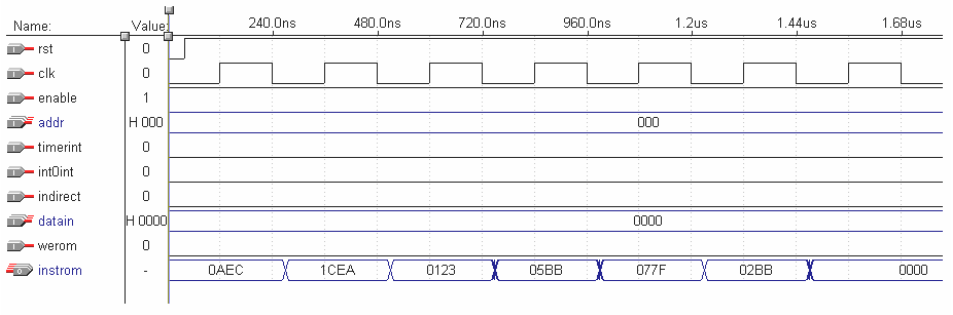
Unit ini akan bekerja bila masukan pin_enable diberi logika *high*. Masukan unit ini dihubungkan ke pin ke-7 dari port A mikrokontroler, sehingga saat unit ini telah diaktifkan maka bila terdeteksi masukan *high*, unit ini akan mengeluarkan sinyal ke unit *program counter*. Hasil simulasi unit ini dapat dilihat pada Gambar 4.9.



Gambar 4.9 Hasil simulasi unit *external interrupt*

4.1.2 Modul ROM

Modul ROM berfungsi untuk menyimpan instruksi mikrokontroler. Pengujian modul ini akan melibatkan unit *program counter* sebagai penunjuk alamat ROM yang akan dikeluarkan. Pengujian dilakukan dengan terlebih dahulu memberikan inisialisasi awal ROM. Hasil simulasi dapat dilihat pada Gambar 4.10.

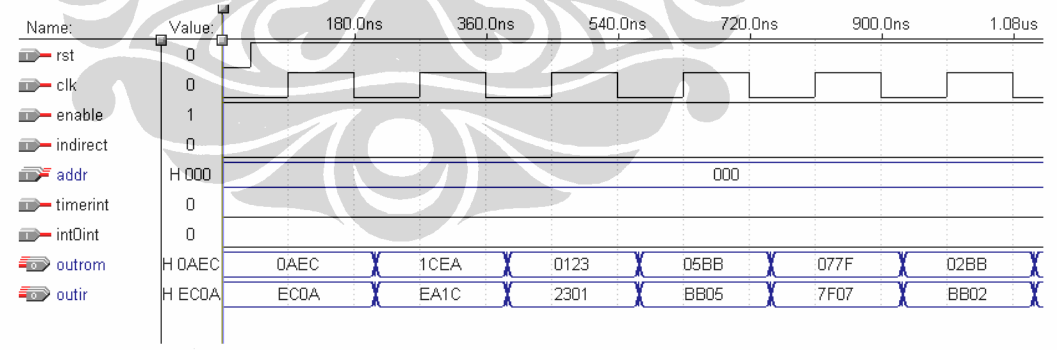


Gambar 4.10 Hasil simulasi unit ROM

Masukan *enable, indirect, addr, timerint,* dan *int0int* merupakan masukan untuk *program counter*. Masukan *datain* dan *werom* merupakan masukan untuk modul ROM. Keluaran *instrom* merupakan keluaran dari modul ROM. Hasil simulasi menunjukkan untuk setiap penambahan penunjukkan alamat dari *program counter*, maka modul ROM akan mengeluarkan isinya berdasarkan alamat yang diberikan oleh *program counter*.

4.1.3 Modul *Instruction Register*

Modul *instruction register* berfungsi untuk menukar *nibble* kode instruksi yang dikeluarkan dari modul ROM sebelum diberikan ke modul *core UIMega 8535*. Hasil simulasi modul ini dapat dilihat pada Gambar 4.11.

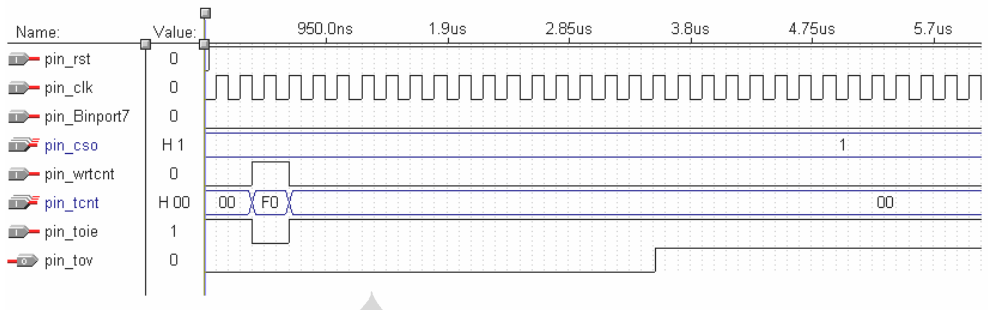


Gambar 4.11 Hasil simulasi modul *instruction register*

Masukan *enable, indirect, addr, timerint,* dan *int0int* merupakan masukan untuk *program counter*. Keluaran *outrom* merupakan keluaran dari modul ROM yang diumpankan ke modul *instruction register*. Keluaran *outir* merupakan keluaran modul *instruction register*.

4.1.4 Modul Timer Interrupt

Modul *timer interrupt* akan mengeluarkan sinyal bilamana hitungan telah mencapai $FF. Hasil pengujian dapat dilihat pada Gambar 4.12.



Gambar 4.12 Hasil simulasi modul *timer interrupt*

Pengujian dilakukan dengan memberikan nilai awal hitungan $F0, kemudian baru diaktifkan. Setelah hitungan telah mencapai $FF maka keluaran *pin_tov* akan bernilai *high*. Pengujian dilakukan dengan menggunakan sinyal *clock* utama sebagai sumber *clock* dengan memberikan *pin_cso* nilai 1.

4.2 VERIFIKASI UIMEGA 8535

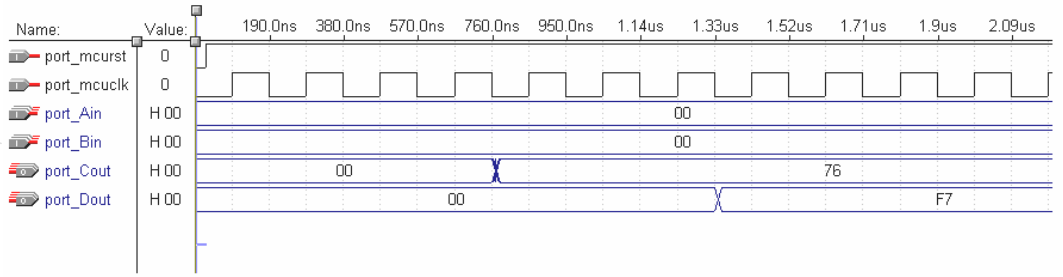
Verifikasi UIMega 8535 dilakukan dengan menggabungkan semua modul dan diuji dengan memberikan sebuah program uji.

4.2.1 Pengujian Dengan Instruksi Aritmatika

Pengujian dilakukan dengan memberikan program uji yang melibatkan instruksi aritmatika. Program uji kemudian menjadi nilai awal dari ROM. Pengujian dilakukan dengan program uji sebagai berikut:

```
; Program uji instruksi aritmatika
ldi r16,$CA     ; isi register 16 dengan data $CA (B 1100 1010)
ldi r17,$AC     ; isi register 17 dengan data $AC (B 1010 1100)
add r16,r17     ; jumlah isi register 16 & 17, hasil disimpan di
                ; register 16
out $15,r16     ; keluarkan isi register 16 ke port C
ldi r18,$80     ; isi register 18 dengan data $80 (B 1000 0000)
adc r16,r18     ; jumlah isi register 16 dan 18 beserta carry, hasil disimpan ;
                ; di register 16
out $12,r16     ; keluarkan isi register 16 ke port D
```

Hasil simulasi:



Gambar 4.13 Hasil uji mikrokontroler dengan instruksi aritmatika

Pengujian melibatkan instruksi aritmatika ADD dan ADC. Instruksi ADD akan melakukan operasi penjumlahan, dan instruksi ADC akan melibatkan nilai *carry* dalam penjumlahan. Pengujian dimulai dengan memberikan UIMega 8535 logika '0' pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian adalah sebagai berikut:

1. Register 16 diisi dengan data $CA atau B 11001010.
2. Register 17 diisi dengan data $AC atau B 10101100.
3. Proses penjumlahan dilakukan dan hasil disimpan di register 16.

   Register 16 **à** $CA = B 1100 1010
   Register 17 **à** $AC = B 1010 1100
   ---------- +
   Register 16 **ß** $76 = B 0111 0110
   carry =1, disimpan di status register bit-0

4. Hasil penjumlahan, yaitu $76, dikeluarkan pada port C yang mempunyai alamat $15.

Proses yang dilakukan oleh pengujian instruksi ADC adalah sebagai berikut:

1. Register 18 diisi dengan data $80 (B 1000 0000).
2. Register 16 telah berisi data $76 (B 0111 0110).
3. Penjumlahan dengan melibatkan *carry* dari proses penjumlahan sebelumnya.

   Register 16 **à** $76 = B 0111 0110
   Register 18 **à** $80 = B 1000 0000
   Carry **à** 1
   ---------- +
   Register 16 **ß** $F7 = B 1111 0111

4. Hasil penjumlahan, yaitu $F7, disimpan pada register 16 dan dikeluarkan pada port D yang mempunyai alamat $12.

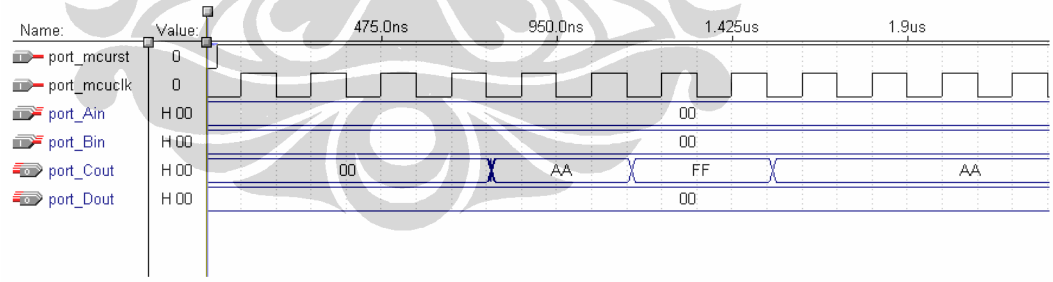
Pengujian dengan melibatkan instruksi aritmatika menunjukkan mikrokontroler telah bekerja dengan baik sebagai satu kesatuan.

4.2.2 Pengujian Dengan Instruksi Percabangan

Pengujian juga dilakukan dengan memberikan program uji yang melibatkan instruksi percabangan. Program uji kemudian menjadi nilai awal dari ROM.

```
; Program uji instruksi percabangan
ldi r16,$AA          ; register 16 diisi data $AA
ldi r17,$FF          ; register 17 diisi data $FF
rcall lompat         ; pemanggilan rutin lompat
out $15,r17          ; isi register 17 dikeluarkan pada port C
rjmp akhir           ; percabangan ke akhir
lompat: out $15,r16  ; isi register 16 dikeluarkan ke port C
ret                  ; kembali ke alamat sebelum pemanggilan rutin
akhir: out $15,r16   ; isi register 16 dikeluarkan ke port C
```

Hasil simulasi:



Gambar 4.14 Hasil uji mikrokontroler dengan instruksi percabangan

Pengujian dilakukan dengan melibatkan instruksi percabangan RJMP dan RCALL. Pengujian dimulai dengan memberikan UIMega 8535 logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian adalah:

1. Register 16 diisi dengan data $AA.
2. Register 17 diisi dengan data $FF.

3. Program kemudian akan memanggil rutin lompat, sehingga eksekusi program dilanjutkan ke alamat rutin lompat. Alamat selanjutnya sebelum pemanggilan rutin dilakukan disimpan ke dalam *stack*.
4. Rutin lompat dijalankan sehingga yang kemudian dilakukan adalah mengeluarkan isi register 16, yaitu $AA, ke port C
5. Instruksi RET akan mengembalikan alamat sebelum pemanggilan rutin dilakukan, sehingga instruksi selanjutnya yang dilakukan adalah *out $15,r17* yang akan mengeluarkan isi register 17, yaitu $FF, ke port C
6. Instruksi RJMP akan melakukan percabangan ke alamat yang ditunjuk oleh label akhir, yaitu merujuk ke instruksi *out $15,r16* yang akan mengeluarkan isi register 16, yaitu $AA, kembali ke port C

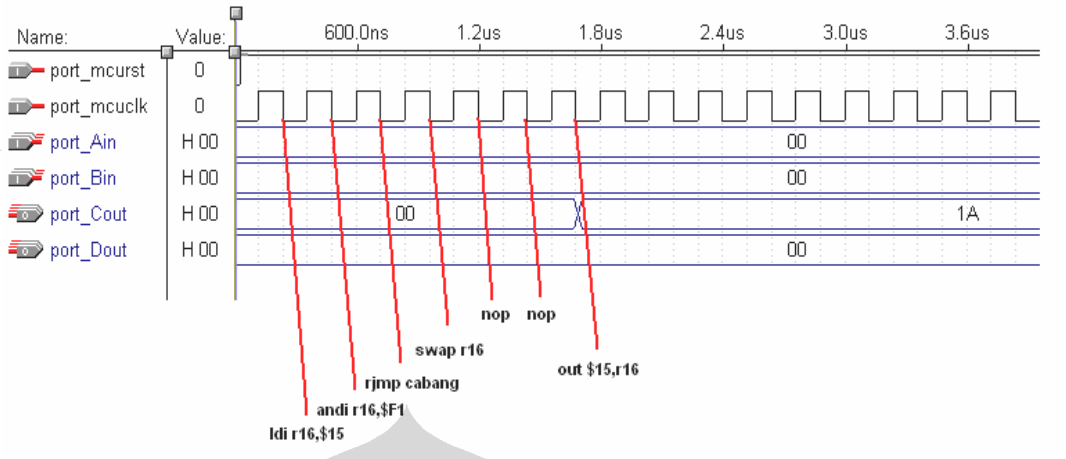
Pengujian dengan melibatkan instruksi aritmatika menunjukkan mikrokontroler telah bekerja dengan baik sebagai satu kesatuan. Pengujian seluruh instruksi ditampilkan pada Lampiran 4.

4.2.3 Pengujian Dengan Kombinasi Instruksi

Pengujian dilakukan tidak hanya dengan satu jenis instruksi. Pengujian dilakukan dengan mengambil salah satu dari beberapa intruksi yang tergabung dalam kelompok instruksi yang sama. Kelompok instruksi perpindahan data (*data transfer instructions*) diambil instruksi LDI dan OUT. Kelompok instruksi aritmatika dan logika diambil instruksi ANDI dan COM. Kelompok instruksi percabangan diambil instruksi RJMP. Kelompok instruksi operasi *bit* diambil instruksi SWAP. Kelompok instruksi kontrol mikrokontroler diambil instruksi NOP. Pengujian dilakukan dengan memberikan instruksi berikut. Program uji kemudian menjadi nilai awal dari ROM.

```
; Program uji beberapa instruksi
ldi r16,$A5     ;B 1010 0101
andi r16,$F1    ;B 1111 0001, hasil: 1010 0001 ($A1)
rjmp cabang     ;lompat ke rutin cabang
com r16         ;operasi komplemen isi register 16
cabang:swap r16       ; operasi pertukaran nibble isi register 16
nop             ;tidak melakukan operasi apapun
nop             ;tidak melakukan operasi apapun
out $15,r16     ;isi register 16 dikeluarkan ke port C
```

Hasil simulasi:



Gambar 4.15 Hasil uji mikrokontroler dengan instruksi kombinasi

Pengujian dimulai dengan memberikan UIMega 8535 logika '0' pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian adalah:

1. Register 16 diisi dengan data $A5 atau B 1010 0101
2. Operasi logika AND dilakukan antara isi register 16 dengan konstanta $F1

   Register 16 à $A5 = B 1100 0101
   $F1 = B 1111 0001
   ---------- AND
   Register 16 ß $A1 = B 1100 0001

3. Percabangan dilakukan ke rutin cabang, sehingga perintah berikutnya yaitu instruksi *COM r16* tidak akan dilakukan.
4. Instruksi di rutin cabang dilakukan, yaitu *SWAP r16*. Instruksi ini akan menukar *nibble* dari isi register 16.

   Register 16 à $A1 = B 1100 0001
   Setelah operasi SWAP: Register 16 ß $1A = B 0001 1100

5. Operasi kontrol *NOP* dilakukan dua siklus sinyal *clock*.
6. Isi dari register 16 dikeluarkan ke port C.

Pengujian dengan melibatkan beberapa jenis instruksi menunjukkan mikrokontroler telah bekerja dengan baik sebagai satu kesatuan. Perbandingan jumlah sinyal *clock* yang dibutuhkan UIMega 8535 dalam melaksanakan instruksi dengan yang dibutuhkan oleh ATMega 8535 dapat dilihat pada Lampiran 3.

BAB V
KESIMPULAN

Beberapa kesimpulan yang dapat diambil dari hasil tesis ini adalah sebagai berikut:

1. Tesis ini telah memberikan kontribusi IP*core* mikrokontroler kompatibel Atmel ATMega 8535 yang dapat mengerti semua instruksi yang dimiliki oleh mikrokontroler Atmel ATMega 8535.
2. Hasil perancangan IP*core* telah disimulasikan dan diverifikasi melalui diagram pewaktuan dengan aplikasi MAX+plus II. Hasil simulasi menunjukkan bahwa UIMega 8535 dapat bekerja dengan baik pada kecepatan sinyal *clock* 4,2 MHz.
3. UIMega 8535 mampu menjalankan hampir semua instruksi ATMega 8535 dalam 1 sinyal *clock*. Instruksi pengambilan (*load*) UIMega 8535 lebih cepat 1 sinyal *clock* daripada ATMega 8535.

DAFTAR ACUAN

[1] Thomas, Eniman Y. Syamsuddin, "Design, Implementation & Verification of Intel 8085 Compatible Microprocessor IPCore in CPLD," *Jurnal Kajian Teknologi*, Universitas Tarumanagara: VI (2) 2004, ISSN: 411-2698, hal. 169-177

[2] Sarwono Sutikno, "Design and implementation of 32-bit RISC Microprocessor Core: a hypothetical microprocessor on FPGA," *Proceedings Industrial Electronic Seminar 1999 (IES'99)*, 1999

[3] Wikipedia (2007). *Microcontroller*. Diakses 20 September 2007, dari Wikimedia Foundation.
http://en.wikipedia.org/wiki/Microcontroller

[4] Me (2002). *What do PIC and AVR mean*. Diakses 14 Oktober 2007, dari Edaboard Forum.
http://www.edaboard.com/ftopic79669.html

[5] Atmel Corporation (2006). *8-bit AVR Microcontroller with 8Kbytes In-System Programmable Flash*. Diakses 19 Agustus 2007, dari Atmel.
www.atmel.com/dyn/resources/prod_document/doc2502.pdf

[6] Atmel Corporation (2005). *8-bit AVR Instruction Set*. Diakses 19 Agustus, dari Atmel.
www.atmel.com/atmel/acrobat/doc0856.pdf

[7] M. M. Mano, *Digital Design* (New Jersey: Prentice Hall, 2002), hal 293

[8] D. Pellerin, D. Taylor, *VHDL Made Easy!* (New Jersey: Prentice Hall,1996), hal. 5

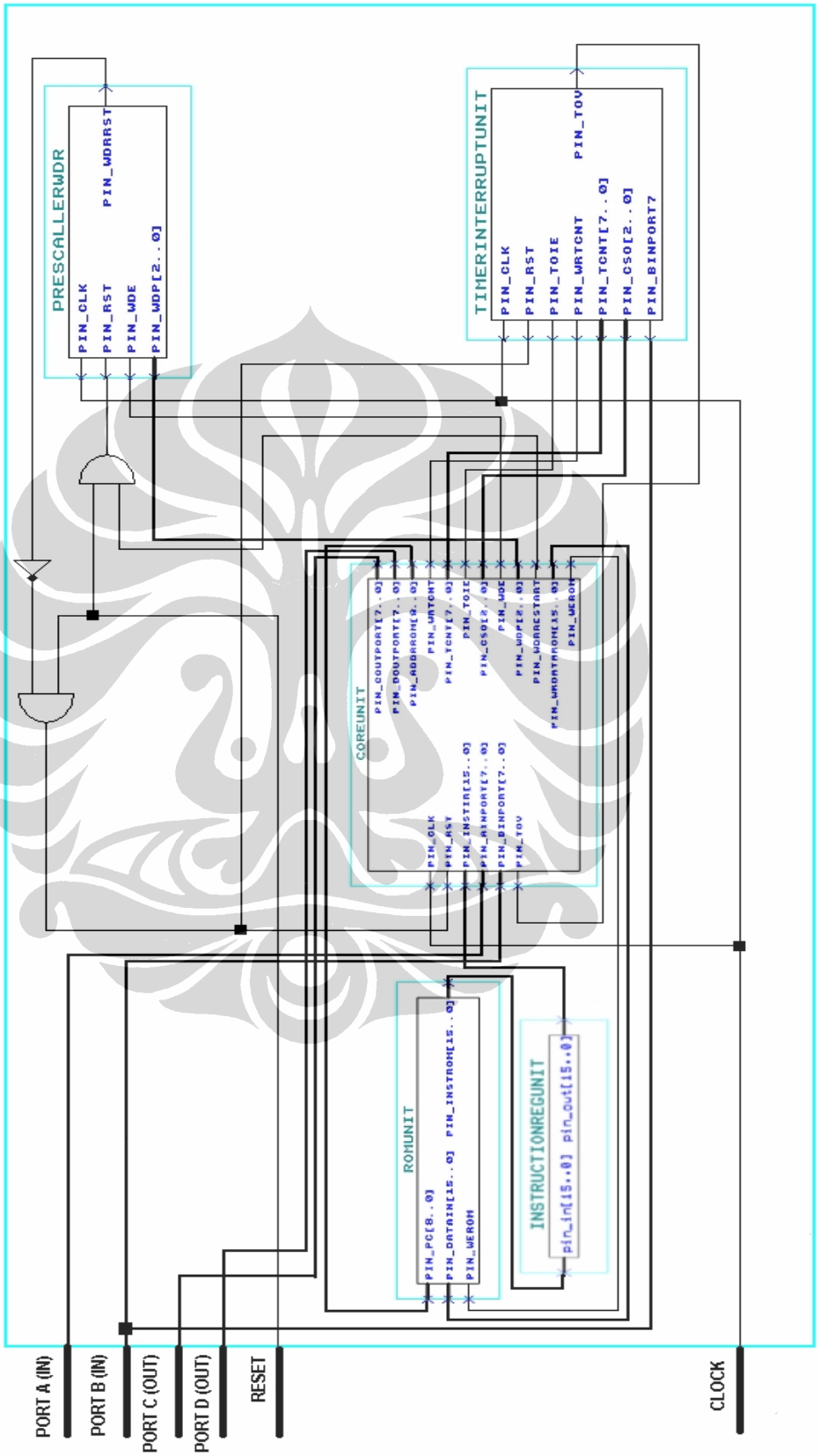
[9] P. J. Ashenden, *The Designer's Guide to VHDL* (San Francisco: Morgan Kaufmann Publishers, 1996), hal. xvii

[10] S. Brown, Z. Vranesic, *Fundamentals of Digital Logic with VHDL Design* (Singapore: McGraw-Hill, 2000), hal. 52

LAMPIRAN 1

HUBUNGAN MODUL-MODUL UIMEGA 8535

HUBUNGAN MODUL-MODUL UIMEGA 8535



LAMPIRAN 2

MNEMONICS DAN 16-BIT KODE INSTRUKSI

MNEMONICS DAN 16-BIT KODE INSTRUKSI

No	Mnemonics	16-Bit Opcode			
1	NOP	0000	0000	0000	0000
2	MOVW	0000	0001	dddd	rrrr
3	MULS	0000	0010	dddd	rrrr
4	MULSU	0000	0011	0ddd	0rrr
5	FMUL	0000	0011	0ddd	1rrr
6	FMULS	0000	0011	1ddd	0rrr
7	FMULSU	0000	0011	1ddd	1rrr
8	CPC	0000	01rd	dddd	rrrr
9	SBC	0000	10rd	dddd	rrrr
10	ADD	0000	11rd	dddd	rrrr
11	LSL	0000	11dd	dddd	dddd
12	CPSE	0001	00rd	dddd	rrrr
13	CP	0001	01rd	dddd	rrrr
14	SUB	0001	10rd	dddd	rrrr
15	ADC	0001	11rd	dddd	rrrr
16	ROL	0001	11dd	dddd	dddd
17	AND	0010	00rd	dddd	rrrr
18	TST	0010	00dd	dddd	dddd
19	EOR	0010	01rd	dddd	rrrr
20	CLR	0010	01dd	dddd	dddd
21	OR	0010	10rd	dddd	rrrr
22	MOV	0010	11rd	dddd	rrrr
23	CPI	0011	KKKK	dddd	KKKK
24	SBCI	0100	KKKK	dddd	KKKK
25	SUBI	0101	KKKK	dddd	KKKK
26	ORI	0110	KKKK	dddd	KKKK
27	SBR	0110	KKKK	dddd	KKKK
28	ANDI	0111	KKKK	dddd	KKKK
29	CBR	0111	KKKK	dddd	KKKK
30	LD (Rd,Y) - Load Indirect	1000	000d	dddd	1000
31	LDD (Rd,Y+q) - Load Indirect With Displacement	10q0	qq0d	dddd	1qqq
32	LD (Rd,Z) - Load Indirect	1000	000d	dddd	0000
33	LDD (Rd,Z+q) - Load Indirect With Displacement	10q0	qq0d	dddd	0qqq
34	ST (Y,Rr) - Store Indirect	1000	001r	rrrr	1000
35	STD (Y+q,Rr) - Store Indirect With Displacement	10q0	qq1r	rrrr	1qqq
36	ST (Z,Rr) - Store Indirect	1000	001r	rrrr	0000
37	STD (Z+q,Rr) - Store Indirect With Displacement	10q0	qq1r	rrrr	0qqq
38	LDS (Rd,k) - Load Direct from SRAM	1001 kkkk	000d kkkk	dddd kkkk	0000 kkkk
39	LD (Rd,Z+) - Load Indirect & Post Inc	1001	000d	dddd	0001
40	LD (Rd,-Z) - Load Indirect & Pre Dec	1001	000d	dddd	0010
41	LPM (Rd,Z) - Load Program Memory	1001	000d	dddd	0100

42	LPM (Rd,Z+) - Load Program Memory & Post Inc	1001	000d	dddd	0101
43	LD (Rd,Y+) - Load Indirect & Post Inc	1001	000d	dddd	1001
44	LD (Rd,-Y) - Load Indirect & Pre Dec	1001	000d	dddd	1010
45	LD (Rd,X) - Load Indirect	1001	000d	dddd	1100
46	LD (Rd,X+) - Load Indirect & Post Inc	1001	000d	dddd	1101
47	LD (Rd,-X) - Load Indirect & Pre Dec	1001	000d	dddd	1110
48	POP	1001	000d	dddd	1111
49	STS (k,Rr) - Store Direct to SRAM	1001 kkkk	001d kkkk	dddd kkkk	0000 kkkk
50	ST (Z+,Rr) - Store Indirect & Post Inc	1001	001r	rrrr	0001
51	ST (-Z,Rr) - Store Indirect & Pre Inc	1001	001r	rrrr	0010
52	ST (Y+,Rr) - Store Indirect & Post Inc	1001	001r	rrrr	1001
53	ST (-Y,Rr) - Store Indirect & Pre Inc	1001	001r	rrrr	1010
54	ST (X,Rr) - Store Indirect	1001	001r	rrrr	1100
55	ST (X+,Rr) - Store Indirect & Post Inc	1001	001r	rrrr	1101
56	ST (-X,Rr) - Store Indirect & Pre Inc	1001	001r	rrrr	1110
57	PUSH	1001	001d	dddd	1111
58	BSET	1001	0100	0sss	1000
59	SEC	1001	0100	0000	1000
60	SEZ	1001	0100	0001	1000
61	SEN	1001	0100	0010	1000
62	SEV	1001	0100	0011	1000
63	SES	1001	0100	0100	1000
64	SEH	1001	0100	0101	1000
65	SET	1001	0100	0110	1000
66	SEI	1001	0100	0111	1000
67	IJMP	1001	0100	0000	1001
68	BCLR	1001	0100	1sss	1000
69	CLC	1001	0100	1000	1000
70	CLZ	1001	0100	1001	1000
71	CLN	1001	0100	1010	1000
72	CLV	1001	0100	1011	1000
73	CLS	1001	0100	1100	1000
74	CLH	1001	0100	1101	1000
75	CLT	1001	0100	1110	1000
76	CLI	1001	0100	1111	1000
77	COM	1001	010d	dddd	0000
78	NEG	1001	010d	dddd	0001
79	SWAP	1001	010d	dddd	0010
80	INC	1001	010d	dddd	0011
81	ASR	1001	010d	dddd	0101

82	LSR	1001	010d	dddd	0110
83	ROR	1001	010d	dddd	0111
84	DEC	1001	010d	dddd	1010
85	RET	1001	0101	0000	1000
86	ICALL	1001	0101	0000	1001
87	RETI	1001	0101	0001	1000
88	SLEEP	1001	0101	1000	1000
89	BREAK	1001	0101	1001	1000
90	WDR	1001	0101	1010	1000
91	LPM - Load Program Memory	1001	0101	1100	1000
92	SPM	1001	0101	1110	1000
93	ADIW	1001	0110	KKdd	KKKK
94	SBIW	1001	0111	KKdd	KKKK
95	CBI	1001	1000	AAAA	Abbb
96	SBIC	1001	1001	AAAA	Abbb
97	SBI	1001	1010	AAAA	Abbb
98	SBIS	1001	1011	AAAA	Abbb
99	MUL	1001	11rd	dddd	rrrr
100	IN	1011	0AAd	dddd	AAAA
101	OUT	1011	1AAr	rrrr	AAAA
102	RJMP	1100	kkkk	kkkk	kkkk
103	RCALL	1101	kkkk	kkkk	kkkk
104	SER	1110	1111	dddd	1111
105	LDI	1110	KKKK	dddd	KKKK
106	BRCS	1111	00kk	kkkk	k000
107	BRLO	1111	00kk	kkkk	k000
108	BREQ	1111	00kk	kkkk	k001
109	BRMI	1111	00kk	kkkk	k010
110	BRVS	1111	00kk	kkkk	k011
111	BRLT	1111	00kk	kkkk	k100
112	BRHS	1111	00kk	kkkk	k101
113	BRTS	1111	00kk	kkkk	k110
114	BRIE	1111	00kk	kkkk	k111
115	BRBS	1111	00kk	kkkk	ksss
116	BRCC	1111	01kk	kkkk	k000
117	BRSH	1111	01kk	kkkk	k000
118	BRNE	1111	01kk	kkkk	k001
119	BRPL	1111	01kk	kkkk	k010
120	BRVC	1111	01kk	kkkk	k011
121	BRGE	1111	01kk	kkkk	k100
122	BRHC	1111	01kk	kkkk	k101
123	BRTC	1111	01kk	kkkk	k110
124	BRID	1111	01kk	kkkk	k111
125	BRBC	1111	01kk	kkkk	ksss
126	BLD	1111	100d	dddd	0bbb
127	BST	1111	101d	dddd	0bbb
128	SBRC	1111	110r	rrrr	0bbb
129	SBRS	1111	111r	rrrr	0bbb

LAMPIRAN 3

PERBANDINGAN JUMLAH CLOCK ATMEGA 8535 DAN UIMEGA 8535

PERBANDINGAN JUMLAH SINYAL CLOCK ANTARA ATMEGA 8535 DENGAN UIMEGA 8535

Mnemonics	#Clock		Mnemonics	#Clock	
	ATMega 8535	UIMega 8535		ATMega 8535	UIMega 8535
ADD	1	1	BREQ	1/2	1
ADC	1	1	BRNE	1/2	1
ADIW	2	1	BRCS	1/2	1
SUB	1	1	BRCC	1/2	1
SUBI	1	1	BRSH	1/2	1
SBC	1	1	BRLO	1/2	1
SBCI	1	1	BRMI	1/2	1
SBIW	2	1	BRPL	1/2	1
AND	1	1	BRGE	1/2	1
ANDI	1	1	BRLT	1/2	1
OR	1	1	BRHS	1/2	1
ORI	1	1	BRHC	1/2	1
EOR	1	1	BRTS	1/2	1
COM	1	1	BRTC	1/2	1
NEG	1	1	BRVS	1/2	1
SBR	1	1	BRVC	1/2	1
CBR	1	1	BRIE	1/2	1
INC	1	1	BRID	1/2	1
DEC	1	1	MOV	1	1
TST	1	1	MOVW	1	1
CLR	1	1	LDI	1	1
SER	1	1	LD	2	1
MUL	2	1	LDD	2	1
MULS	2	1	ST	2	3
MULSU	2	1	STD	2	3
FMUL	2	1	STS	2	4
FMULS	2	1	LDS	2	2
FMULSU	2	1	LPM	3	2
RJMP	2	1	IN	1	1
IJMP	2	1	OUT	1	1
RCALL	3	1	PUSH	2	1
ICALL	3	1	POP	2	1
RET	4	1	SBI	2	1
RETI	4	1	CBI	2	1
CPSE	1/2/3	2	LSL	1	1
CP	1	1	LSR	1	1
CPC	1	1	ROL	1	1
CPI	1	1	ROR	1	1
SBRC	1/2/3	2	ASR	1	1
SBRS	1/2/3	2	SWAP	1	1
SBIC	1/2/3	2	BSET	1	1
SBIS	1/2/3	2	BCLR	1	1
BRBS	1/2	1	BST	1	1
BRBC	1/2	1	BLD	1	1
SEC	1	1	CLC	1	1
SEN	1	1	CLN	1	1
SEZ	1	1	CLZ	1	1
SEI	1	1	CLI	1	1
SES	1	1	CLS	1	1
SEV	1	1	CLV	1	1
SET	1	1	CLT	1	1
SHE	1	1	CLH	1	1
NOP	1	1	WDR	1	1
SLEEP	1	1	BREAK	1	1
SPM	4	4			

LAMPIRAN 4

PENGUJIAN INSTRUKSI

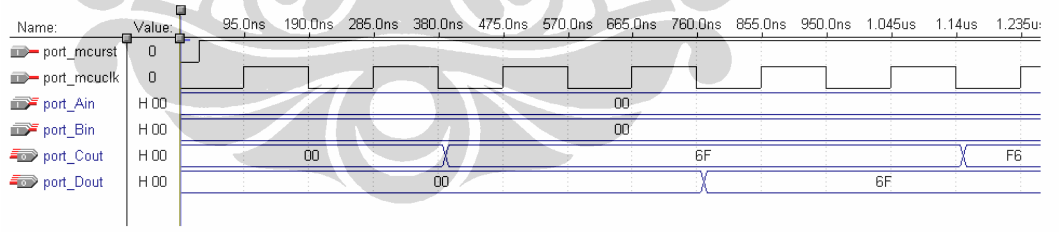
1. PENGUJIAN INSTRUKSI LDI, MOV, SWAP, OUT

Instruksi LDI (*Load Immediate*) mempunyai bentuk *LDI Rd,K*. Instruksi ini akan menyalin data K ke register Rd (*Rd ß K*). Instruksi MOV (*Copy Register*) mempunyai bentuk *MOV Rd,Rr*. Instruksi ini akan menyalin isi register Rr ke register Rd (*Rd ß Rr*). Instruksi SWAP (*Swap Nibbles*) mempunyai bentuk *SWAP Rd*. Instruksi ini akan menukar *nibble* isi dari register Rd (*Rd[3..0] ß Rd[7..4],Rd[7..4] ß Rd[3..0]*). Instruksi OUT (*Store Register to I/O Location*) mempunyai bentuk *OUT P,Rr*. Instruksi ini akan mengirim isi register Rr ke I/O port (*P ß Rr*). Port C memiliki alamat $15 dan port D memiliki alamat $12.

Pengujian instruksi LDI, MOV, SWAP, dan OUT dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi LDI, MOV, SWAP, OUT
ldi r16,$6F     ; isi register 16 dengan data $6F
out $15,r16     ; keluarkan isi register 16 ke port C
mov r17,r16     ; pindahkan isi register 16 ke register 17
out $12,r17     ; keluarkan isi register 17 ke port D
swap r17        ; tukar nibble isi register 17
out $15,r17     ; keluarkan isi register 17 ke port C
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Instruksi *ldi r16,$6F* akan menyimpan data $6F ke register 16. Kemudian isi register ini akan dikeluarkan pada port C. Instruksi berikutnya adalah *mov r17,r16* akan menyalin isi register 17 ke register 16. Kemudian isi register 17 akan dikeluarkan pada port D. Terakhir adalah instruksi *swap r17*. Instruksi ini akan menukar *nibble* dari isi register 17, yaitu $6F menjadi $F6. Kemudian isi register ini dikeluarkan pada port C.

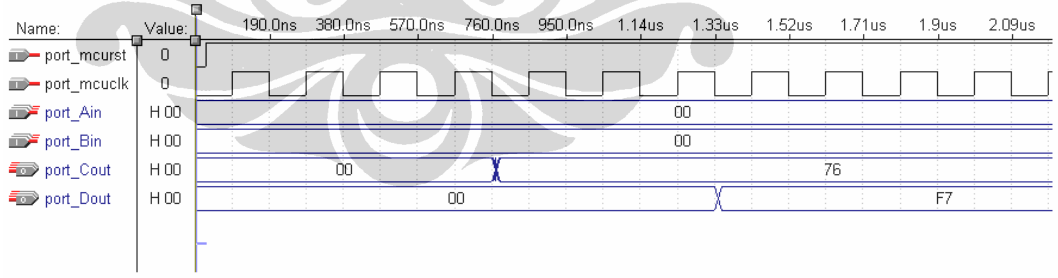
2. PENGUJIAN INSTRUKSI ADD, ADC

Instruksi ADD (*Add two Registers*) mempunyai bentuk *ADD Rd,Rr*. Instruksi ini akan melakukan operasi aritmatik penjumlahan isi register Rd dan isi register Rr, dengan hasil akan disimpan pada register Rd (*Rd ß Rd+Rr*). Instruksi ADC (*Add with Carry Two Registers*) mempunyai bentuk *ADC Rd,Rr*. Instruksi ini akan melakukan operasi aritmatik penjumlahan isi register Rd dan isi register Rr sekaligus *carry* dari *status register*, dengan hasil akan disimpan pada register Rd (*Rd ß Rd+Rr+C*).

Pengujian instruksi ADD dan ADC dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi ADD, ADC
ldi r16,$CA     ; isi register 16 dengan data $CA (B 1100 1010)
ldi r17,$AC     ; isi register 17 dengan data $AC (B 1010 1100)
add r16,r17     ; jumlah isi register 16 & 17, hasil disimpan di
                ; register 16
out $15,r16     ; keluarkan isi register 16 ke port C
ldi r18,$80     ; isi register 18 dengan data $80 (B 1000 0000)
adc r16,r18     ; jumlah isi register 16 dan 18 beserta carry, hasil disimpan ;
                ; di register 16
out $12,r16     ; keluarkan isi register 16 ke port D
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi ADD adalah sebagai berikut.

5. Register 16 diisi dengan data $CA atau B 11001010.
6. Register 17 diisi dengan data $AC atau B 10101100.
7. Proses penjumlahan dilakukan dan hasil disimpan di register 16.

Register 16 à $CA = B 1100 1010
Register 17 à $AC = B 1010 1100
---------- +
Register 16 ß $76 = B 0111 0110
carry =1, disimpan di status register bit-0

8. Hasil penjumlahan, yaitu $76, dikeluarkan pada port C yang mempunyai alamat $15.

Proses yang dilakukan oleh pengujian instruksi ADC adalah sebagai berikut:

5. Register 18 diisi dengan data $80 (B 1000 0000).
6. Register 16 telah berisi data $76 (B 0111 0110).
7. Penjumlahan dengan melibatkan *carry* dari proses penjumlahan sebelumnya.

Register 16 à $76 = B 0111 0110
Register 18 à $80 = B 1000 0000
Carry à 1
---------- +
Register 16 ß $F7 = B 1111 0111

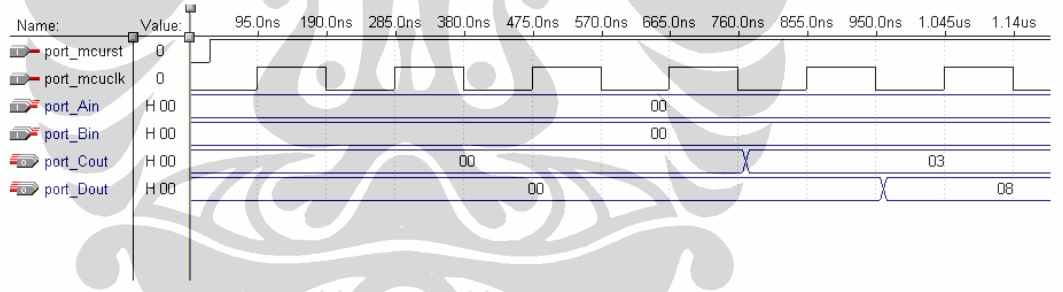
8. Hasil penjumlahan, yaitu $F7, disimpan pada register 16 dan dikeluarkan pada port D yang mempunyai alamat $12.

3. PENGUJIAN INSTRUKSI ADIW

Instruksi ADIW (*Add Immediate to Word*) mempunyai bentuk *ADIW Rdl,K*. Instruksi ini akan melakukan operasi aritmatik penjumlahan antara nilai konstan K dengan isi pasangan register dan hasil operasi akan disimpan pada pasangan register tersebut (*Rdh:Rdl ß Rdh:Rdl+K*). Pasangan register yang dipakai adalah register 24 dan 25, register 26 dan 27, register 28 dan 29, register 30 dan 31. Pengujian instruksi ADIW dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi ADIW
ldi r24,$02      ; isi register 24 dengan data $02
ldi r25,$07      ; isi register 25 dengan data $07
adiw r24,1       ; penjumlahan isi register 24 dengan 1 dan isi register 25 dengan 1,
                 ; hasil disimpan kembali pada register bersangkutan
out $15,r24      ; keluarkan isi register 24 ke port C
out $12,r25      ; keluarkan isi register 25 ke port D
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi ADIW adalah sebagai berikut.

1. Register 24 diisi dengan data $02.
2. Register 25 diisi dengan data $07.
3. Instruksi ADIW akan melakukan penjumlahan ke pasangan register 24 dan 25 dengan 1 dan hasil akan disimpan kembali pada register bersangkutan.
4. Isi register 24, yaitu $03 yang merupakan hasil operasi, dikeluarkan pada port C.
5. Isi register 25, yaitu $08 yang merupakan hasil operasi, dikeluarkan pada port D.

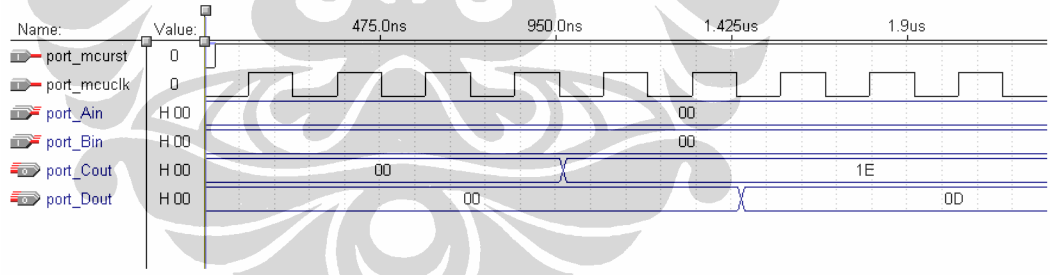
4. PENGUJIAN INSTRUKSI SUB, SUBI

Instruksi SUB (*Subracts Two Registers*) mempunyai bentuk *SUB Rd,Rr*. Instruksi ini akan melakukan operasi aritmatika pengurangan, yaitu antara isi register Rd dengan isi register Rr, kemudian hasil operasi akan disimpan di register Rd (*Rd ß Rd-Rr*). Instruksi SUBI (*Subract Constant from Register*) mempunyai bentuk *SUBI Rd,K*. Instruksi ini akan melakukan operasi aritmatika pengurangan, yaitu antara isi register Rd dengan konstanta K (*Rd ß Rd-K*).

Pengujian instruksi SUB dan SUBI dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi SUB, SUBI
ldi r16,$CA     ; isi register 16 dengan data $CA
ldi r17,$AC     ; isi register 17 dengan data $AC
sub r16,r17     ; lakukan operasi pengurangan isi register 16 dengan isi register 17
                ; dan hasil disimpan di register 16
out $15,r16     ; keluarkan isi register 16 ke port C
subi r16,$11    ; lakukan operasi pengurangan isi register 16 dengan nilai $11
out $12,r16     ; keluarkan isi register 16 ke port D
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi SUB adalah sebagai berikut.

1. Register 16 diisi dengan data $CA atau B 11001010.
2. Register 17 diisi dengan data $AC atau B 10101100.
3. Proses penjumlahan dilakukan dan hasil disimpan di register 16.

   Register 16 à $CA = B 1100 1010
   Register 17 à $AC = B 1010 1100
   ---------- -
   Register 16 ß $1E = B 0001 1110

4. Hasil penjumlahan, yaitu $1E, dikeluarkan pada port C yang mempunyai alamat $15.

Proses yang dilakukan oleh pengujian instruksi SUBI adalah sebagai berikut:

1. Register 16 telah terisi dengan data $1E atau B 0001 1110.
2. instruksi SUBI akan melakukan operasi pengurangan isi register 16 dengan nilai $11.
3. Proses pengurangan dilakukan dan hasil disimpan di register 16.

Register 16 à $1E = B 0001 1110
$11 = B 0001 0001
---------- -
Register 16 ß $0D = B 0000 1101

4. Hasil penjumlahan, yaitu $0D, dikeluarkan pada port D yang mempunyai alamat $12.

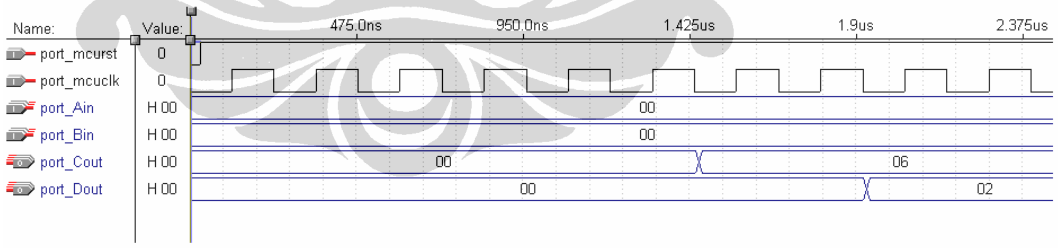
5. PENGUJIAN INSTRUKSI SBC DAN SBCI

Instruksi SBC (*Subract with Carry Two Registers*) mempunyai bentuk *SBC Rd,Rr*. Instruksi ini akan melakukan operasi aritmatik pengurangan antara isi register Rd dengan isi register Rr dan *carry* yang tersimpan di *status register*, dan kemudian hasil operasi akan disimpan kembali di register Rd (*Rd ß Rd-Rr-Carry*). Instruksi SBCI (*Subract with Carry Constant from Register*) mempunyai bentuk *SBCI Rd,K*. Instruksi ini akan melakukan operasi aritmatik pengurangan sebuah bilangan K dan *carry* yang tersimpan di *status register*, dan kemudian hasil operasi akan disimpan kembali di register Rd (*Rd ß Rd-K-Carry*).

Pengujian instruksi SBC dan SBCI dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi SBC dan SBCI
ldi r18,$01     ; register 18 diisi data $01 atau B 0000 0001
out $3F,r18     ; set carry di status register
ldi r16,$09     ; register 16 diisi data $09
ldi r17,$02     ; register 17 diisi data $02
sbc r16,r17     ; subtract with carry antara register 16 dan register 17
out $15,r16     ; isi register 16 dikeluarkan ke port C
sbci r16,$04    ; subract with carry constant antara isi register 16 dan $04
out $12,r16     ; isi register 16 dikeluarkan ke port D
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi SBC adalah sebagai berikut.

1. Register 18 diisi dengan data $01 atau B 0000 0001.
2. Isi register 18 dikeluarkan ke *status register* dengan alamat $3F, dengan demikian flag *carry* diberi logika ’1’.

3. Register 16 diisi dengan data $09.
4. Register 17 diisi dengan data $02.
5. Instruksi SBC akan melakukan operasi sebagai berikut:

Register 16 à $09 = B 0000 1001
Register 17 à $02 = B 0000 0010
Carry à 1
---------- -
Register 16 ß $06 = B 0000 0110
Carry ß 0

6. Hasil dari operasi disimpan di register 16 dan dikeluarkan ke port C dengan alamat $15.

Proses dilanjutkan dengan intruksi SBCI antara isi register 16 ($06) dengan konstanta $04, yaitu:

Register 16 à $06 = B 0000 0110
Konstanta à $04 = B 0000 0100
Carry à 0
---------- -
Register 16 ß $02 = B 0000 0010

Hasil operasi, yaitu isi register 16, dikeluarkan pada port D dengan alamat $12.

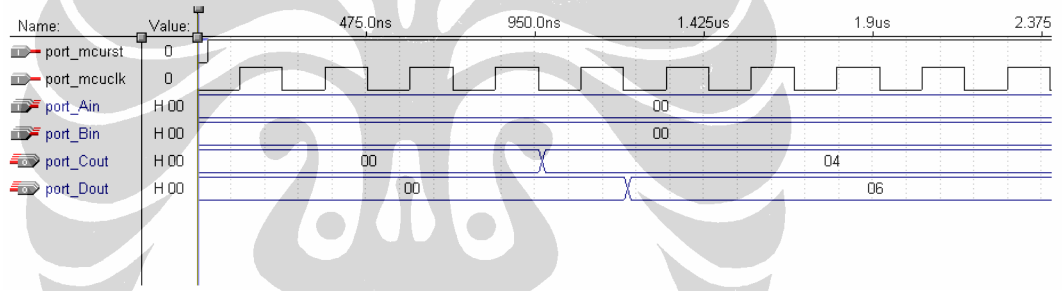
6. PENGUJIAN INSTRUKSI SBIW

Instruksi SBIW (*Subract Immediate from Word*) mempunyai bentuk *SBIW Rdl,K*. Instruksi ini akan melakukan pengurangan isi register Rd *high byte* dan Rd *low byte* dengan konstanta K (*Rdh:Rdl ß Rdh:Rdl-K*).

Pengujian instruksi SBIW dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi SBIW
ldi r24,$05      ; register 24 diisi dengan data $05
ldi r25,$07      ; register 25 diisi dengan data $07
sbiw r24,1       ; isi register 24 dan register 25 dikurang dengan 1
out $15,r24      ; isi register 24 dikeluarkan pada port C
out $12,r25      ; isi register 25 dikeluarkan pada port D
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi SBIW adalah sebagai berikut:

1. Register 24 diisi dengan data $05.
2. Register 25 diisi dengan data $07.
3. Isi register 24 dan 25 dikurang dengan konstanta 1 dan hasil disimpan kembali pada register bersangkutan, yaitu $04 pada register 24, dan $06 pada register 25.
4. Isi register 24 dikeluarkan pada port C dengan alamat $15.
5. Isi register 25 dikeluarkan pada port D dengan alamat $12.

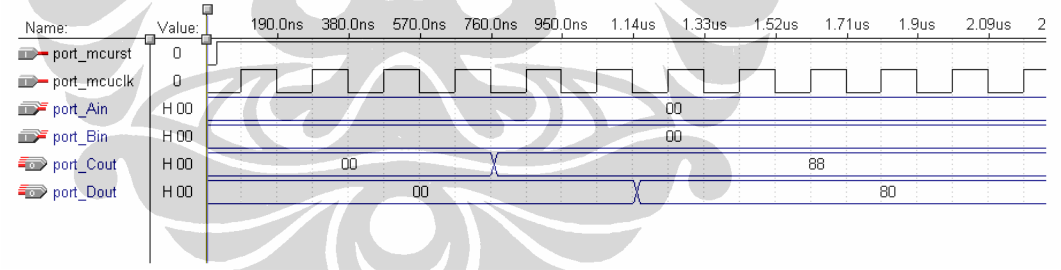
7. PENGUJIAN INSTRUKSI AND DAN ANDI

Instruksi AND (*Logical AND Registers*) mempunyai bentuk *AND Rd,Rr*. Instruksi ini akan melakukan operasi logika AND terhadap isi register Rd dan Rr. Hasil operasi akan disimpan kembali pada register Rd (*Rd ß Rd • Rr*). Instruksi ANDI (*Logical AND Registers and Constant*) mempunyai bentuk *ANDI Rd,K.* Instruksi ini akan melakukan operasi logika AND terhadap isi register Rd dan konstanta K. Hasil operasi akan disimpan kembali pada register Rd (*Rd ß Rd•K*).

Pengujian instruksi AND dan ANDI dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi AND dan ANDI
ldi r16,$CA     ; register 16 diisi dengan data $CA atau B 1100 1010
ldi r17,$AC     ; register 17 diisi dengan data $AC atau B 1010 1100
and r16,r17     ; operasi logika AND antara isi register 16 dan register 17
out $15,r16     ; isi register 16 dikeluarkan pada port C
andi r16,$F7    ; operasi logika AND antara isi register 16 dan konstanta $F7 atau
                ; B 1111 0111
out $12,r16     ; isi register 16 dikeluarkan pada port D
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi AND dan ANDI adalah sebagai berikut:

1. Register 16 diisi dengan data $CA.
2. Register 17 diisi dengan data $AC.
3. Isi register 16 dan 17 dilakukan operasi AND. Hasil operasi disimpan kembali pada register 16.

Register 16 à $CA = B 1100 1010
Register 17 à $AC = B 1010 1100
---------- AND
Register 16 ß $88 = B 1000 1000

4. Isi register 16 dikeluarkan pada port C.
5. Isi register 16 dilakukan operasi AND dengan konstanta $F7. Hasil operasi disimpan kembali pada register 16.

Register 16 à $88 = B 1000 1000
$F7 = B 1111 0111
---------- AND
Register 16 ß $80 = B 1000 0000

6. Isi register 16 dikeluarkan pada port D.

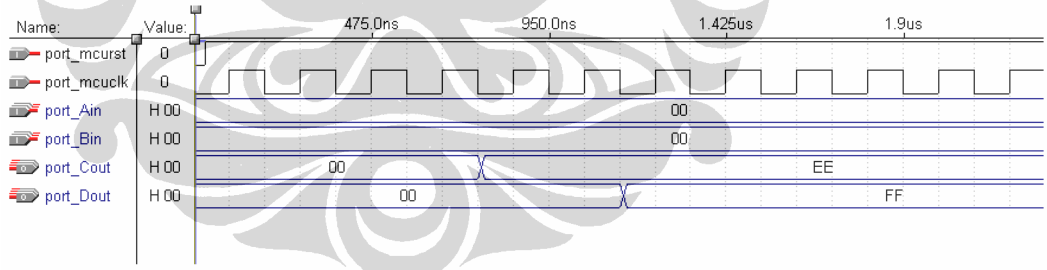
8. PENGUJIAN INSTRUKSI OR DAN ORI

Instruksi OR (*Logical OR Registers*) mempunyai bentuk *OR Rd,Rr*. Instruksi ini akan melakukan operasi logika OR terhadap isi dari register Rd dan Rr dan kemudian hasil operasi kembali disimpan ke register Rd (*Rd ß Rd v Rr*). Instruksi ORI (*Logical OR Registers and Contant*) mempunyai bentuk *ORI Rd,K* akan melakukan operasi logika OR terhadap isi register Rd terhadap konstanta K dan hasil operasi kembali disimpan di register Rd (*Rd ß Rd v K*).

Pengujian instruksi OR dan ORI dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi OR dan ORI
ldi r16,$CA     ; register 16 diisi data $CA atau B 1100 1010
ldi r17,$AC     ; register 17 diisi data $AC atau B 1010 1100
or r16,r17      ; operasi logika OR terhadap isi register 16 dan 17
                ; hasil disimpan di register 16
out $15,r16     ; isi register 16 dikeluarkan pada port C
ori r16,$11     ; operasi logika OR antara isi register 16 dan konstanta $11
                ; hasil disimpan di register 16
out $12,r16     ; isi register 16 dikeluarkan ke port D
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi OR dan ORI adalah sebagai berikut:

1. Register 16 diisi dengan data $CA.
2. Register 17 diisi dengan data $AC.
3. Isi register 16 dan 17 dilakukan operasi OR. Hasil operasi disimpan kembali pada register 16.

Register 16 à $CA = B 1100 1010
Register 17 à $AC = B 1010 1100
---------- OR
Register 16 ß $EE = B 1110 1110

4. Isi register 16 dikeluarkan pada port C.
5. Isi register 16 dilakukan operasi ORI dengan konstanta $11. Hasil operasi disimpan kembali pada register 16.

Register 16 à $EE = B 1110 1110
$11 = B 0001 0001
---------- ORI
Register 16 ß $FF = B 1111 1111

6. Isi register 16 dikeluarkan pada port D.

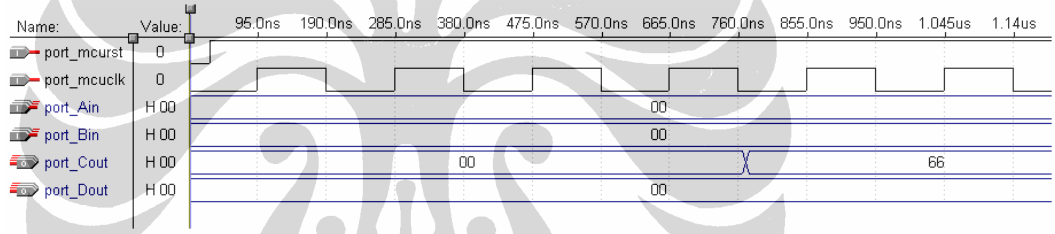
9. PENGUJIAN INSTRUKSI EOR

Instruksi EOR (*Exclusive OR Registers*) mempunyai bentuk *EOR Rd,Rr*. Instruksi ini akan melakukan operasi *Exclusive-OR* terhadap isi register Rd dan isi register Rr. Hasil operasi akan disimpan kembali ke register Rd (*Rd ß Rd⊕ Rr*).

Pengujian instruksi EOR dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi EOR
ldi r16,$CA     ; Register 16 diisi dengan data $CA
ldi r17,$AC     ; Register 17 diisi dengan data $AC
eor r16,r17     ; Operasi Ex-OR terhadap isi register 16 dan isi register 17
                ; hasil disimpan kembali pada register 16
out $15,r16     ; Isi register 16 dikeluarkan pada port C
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi EOR adalah sebagai berikut:

1. Register 16 diisi dengan data $CA.
2. Register 17 diisi dengan data $AC.
3. Isi register 16 dan 17 dilakukan operasi OR. Hasil operasi disimpan kembali pada register 16.

   Register 16 à $CA = B 1100 1010
   Register 17 à $AC = B 1010 1100
   ---------- EOR
   Register 16 ß $66 = B 0110 0110

4. Isi register 16 dikeluarkan pada port C.

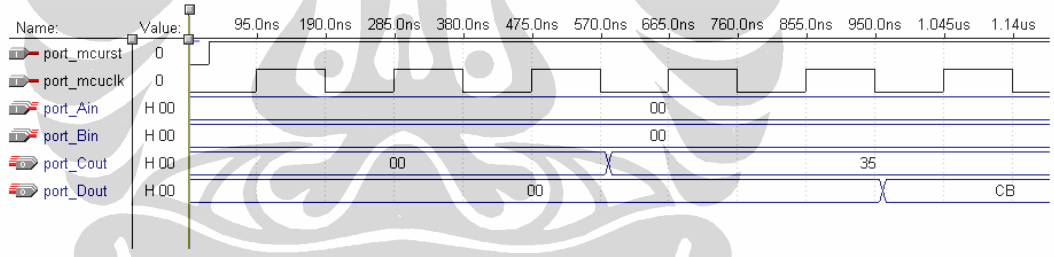
10. PENGUJIAN INSTRUKSI COM DAN NEG

Instruksi COM (*One's Complement*) mempunyai bentuk *COM Rd*. Instruksi ini akan melakukan operasi *One's complement* pada isi register Rd dan hasil kembali disimpan pada register Rd (*Rd ß 0xFF-Rd*). Instruksi NEG (*Two's Complement*) mempunyai bentuk *NEG Rd* akan melakukan operasi *Two's complement* pada isi register Rd dan hasil kembali disimpan pada register Rd (*Rd ß 0x00-Rd*).

Pengujian instruksi COM dan NEG dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi COM dan NEG
ldi r16,$CA     ; register 16 diisi data $CA atau B 1100 1010
com r16         ; operasi one's complement isi register 16
out $15,r16     ; isi register 16 dikeluarkan pada port C
neg r16         ; operasi two's complement isi register 16
out $12,r16     ; isi register 16 dikeluarkan pada port D
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika '0' pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi COM dan NEG adalah sebagai berikut:

1. Register 16 diisi dengan data $CA.
2. Operasi *One's complement* dilakukan terhadap isi register 16. Operasi ini akan melakukan pembalikkan *bit* terhadap isi register 16. Hasil kembali disimpan pada register 16.

Register 16 à $CA = B 1100 1010
---------- NOT
Register 16 ß $35 = B 0011 0101

3. Isi register 16 dikeluarkan pada port C dengan alamat $15.
4. Operasi *Two's complement* dilakukan terhadap isi register 16. Operasi ini akan melakukan pembalikkan *bit* terhadap isi register 16 kemudian ditambah dengan 1. Hasil kembali disimpan pada register 16.

   Register 16 à $35 = B 0011 0101
   ---------- NOT
   = B 1100 1010
   1
   ---------- +
   Register 16 ß $CB = B 1100 1011

5. Isi register 16 dikeluarkan pada port D dengan alamat $12.

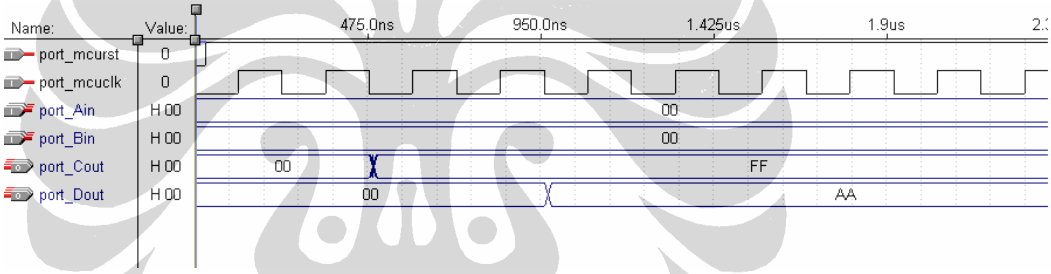
11. PENGUJIAN INSTRUKSI SBR DAN CBR

Instruksi SBR (*Set Bit(s) in Register*) mempunyai bentuk *SBR Rd,K*. Instruksi ini akan memberikan logika '1' terhadap isi register Rd *bit* ke K (*Rd ß Rd v K*). Instruksi CBR (*Clear Bit(s) in Register*) mempunyai bentuk *CBR Rd,K*. Instruksi ini akan melakukan sebaliknya (*Rd ß Rd•(0xFF-K)*).

Pengujian instruksi SBR dan CBR dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi SBR dan CBR
sbr r16,$FF     ; set bit untuk high dan low nibble isi register 16
out $15,r16     ; isi register 16 dikeluarkan ke port C
cbr r16,$55     ; clear bit untuk bit ke-0,2,4, dan 6
out $12,r16     ; isi register 16 dikeluarkan ke port D
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika '0' pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi SBR dan CBR adalah sebagai berikut:

1. Isi register 16 diberi logika '1'. $FF melambangkan semua *bit* posisi dari isi register 16.
2. Isi register 16 dikeluarkan ke portd C dengan alamat $15.
3. Isi register 16 diberi logika '0' pada *bit* ke-0, 2, 4, dan 6. $55 atau B 01010101 melambangkan posisi *bit* ke-0, 2, 4, dan 6, sehingga menghasilkan $AA atau B 1010 1010.
4. Kemudian isi register 16 dikeluarkan pada port D dengan alamat $12.

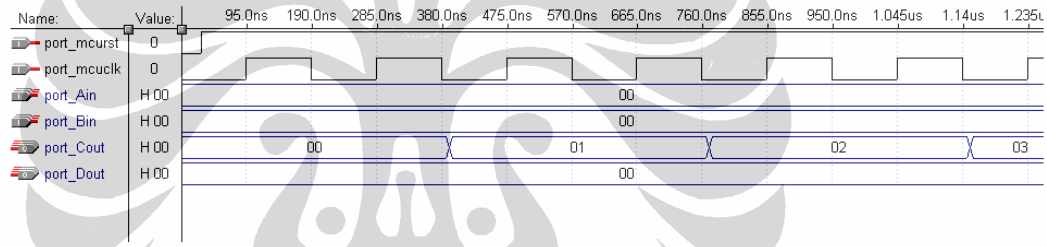
12. PENGUJIAN INSTRUKSI INC

Instruksi INC (*Increment*) mempunyai bentuk *INC Rd*. Instruksi ini akan melakukan operasi penambahan 1 terhadap isi register Rd kemudian hasil operasi disimpan kembali pada register Rd (*Rd ß Rd+1*).

Pengujian instruksi INC dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi INC
ldi r16,$01      ; register 16 diisi dengan data $01
out $15,r16      ; isi register 16 dikeluarkan ke port C
inc r16          ; isi register 16 ditambah 1
out $15,r16      ; isi register 16 dikeluarkan ke port C
inc r16          ; isi register 16 ditambah 1
out $15,r16      ; isi register 16 dikeluarkan ke port C
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi INC adalah sebagai berikut:

1. Register 16 diisi dengan data $01.
2. Isi register 16 dikeluarkan pada port C.
3. Isi register 16 ditambah 1 dan hasil disimpan kembali pada register 16.
4. Isi register 16 dikeluarkan pada port C.
5. Isi register 16 ditambah 1 dan hasil disimpan kembali pada register 16.

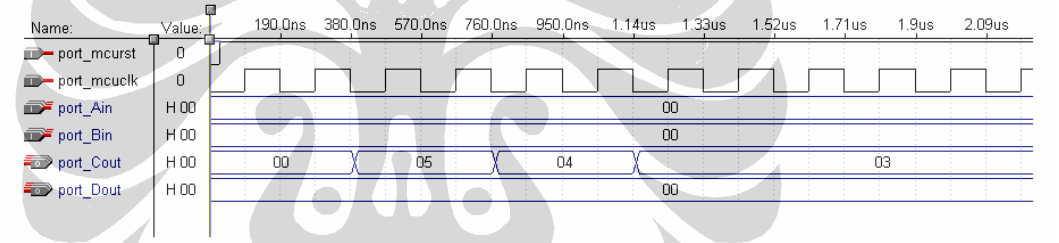
13. PENGUJIAN INSTRUKSI DEC

Instruksi DEC (*Decrement*) mempunyai bentuk *DEC Rd.* Instruksi ini akan melakukan operasi pengurangan 1 terhadap isi register Rd kemudian hasil operasi disimpan kembali pada register Rd (*Rd ß Rd-1*).

Pengujian instruksi DEC dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi DEC
ldi r16,$05      ; register 16 diisi data $05
out $15,r16      ; isi register 16 dikeluarkan pada port C
dec r16          ; operasi decrement isi register 16
out $15,r16      ; isi register 16 dikeluarkan pada port C
dec r16          ; operasi decrement isi register 16
out $15,r16      ; isi register 16 dikeluarkan pada port C
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi DEC adalah sebagai berikut:

1. Register 16 diisi dengan data $05.
2. Isi register 16 dikeluarkan pada port C.
3. Isi register 16 dikurang 1 dan hasil disimpan kembali pada register 16.
4. Isi register 16 dikeluarkan pada port C.
5. Isi register 16 dikurang 1 dan hasil disimpan kembali pada register 16.

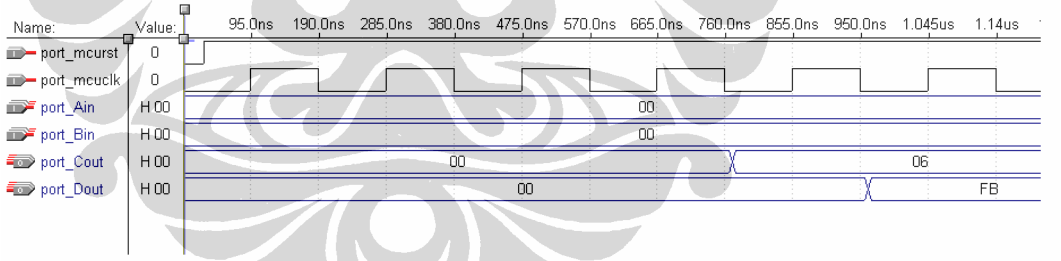
14. PENGUJIAN INSTRUKSI MUL

Instruksi MUL (*Multiply Unsigned*) mempunyai bentuk *MUL Rd,Rr*. Instruksi ini akan melakukan operasi aritmatik perkalian tak bertanda antara isi register Rd dengan isi register Rr (*R1:R0 ß RdxRr*). Operasi ini melakukan perkalian antara *multiplicant* 8-*bit* tak bertanda dengan *multiplier* 8-*bit* tak bertanda dan menghasilkan 16-*bit* hasil operasi. Hasil operasi ini akan disimpan pada register 16 untuk *low byte* dan register 17 untuk *high byte*.

Pengujian instruksi MUL dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi MUL
ldi r20,$FE     ; register 20 diisi data $FE
ldi r21,$FD     ; register 21 diisi data $FD
mul r20,r21     ; operasi MUL antara isi register 20 dan isi register 21
                ; hasil disimpan pada register 16 dan register 17
out $15,r16     ; isi register 16 dikeluarkan pada port C
out $12,r17     ; isi register 17 dikeluarkan pada port D
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi MUL adalah sebagai berikut:

1. Register 20 diisi dengan data $FE atau 1111 1110 (biner) atau 254 (desimal).
2. Register 21 diisi dengan data $FD atau 1111 1101 (biner) atau 253 (desimal).
3. Isi register 20 dan isi register 21 dilakukan operasi perkalian dan hasil disimpan pada register 16 (*low byte*) dan register 17 (*high byte*).

Register 20 à $FE = 254 (desimal)
Register 21 à $FD = 253 (desimal)
---------- x (unsigned)
$FB06 = 64262 (desimal) ;

Register 17 ß $FB, Register 16 ß $06

4. Isi register 16 dikeluarkan pada port C.
5. Isi register 17 dikeluarkan pada port D.

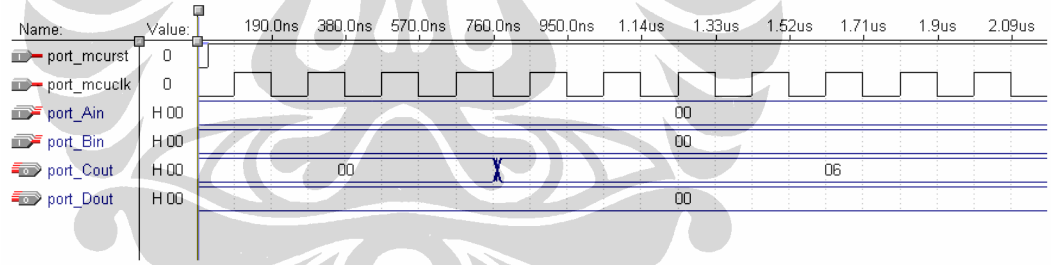
15. PENGUJIAN INSTRUKSI MULS

Instruksi MULS (*Multiply Signed*) mempunyai bentuk *MULS Rd,Rr*. Instruksi ini akan melakukan operasi aritmatik perkalian bertanda antara isi register Rd dengan isi register Rr (*R1:R0 ß RdxRr*). Operasi ini melakukan perkalian antara *multiplicant* 8-*bit* bertanda dengan *multiplier* 8-*bit* bertanda dan menghasilkan 16-*bit* hasil operasi. Hasil operasi ini akan disimpan pada register 16 untuk *low byte* dan register 17 untuk *high byte.*

Pengujian instruksi MULS dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi MULS
ldi r20,$FE     ; register 20 diisi data $FE
ldi r21,$FD     ; register 21 diisi data $FD
muls r20,r21    ; operasi MULS antara isi register 20 dan isi register 21
                ; hasil disimpan pada register 16 dan register 17
out $15,r16     ; isi register 16 dikeluarkan pada port C
out $12,r17     ; isi register 17 dikeluarkan pada port D
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi MULS adalah sebagai berikut:

1. Register 20 diisi dengan data $FE atau 1111 1110 (biner) atau -2 (desimal).
2. Register 21 diisi dengan data $FD atau 1111 1101 (biner) atau -3 (desimal).
3. Isi register 20 dan isi register 21 dilakukan operasi perkalian dan hasil disimpan pada register 16 (*low byte*) dan register 17 (*high byte*).

Register 20 à $FE = -2 (desimal)
Register 21 à $FD = -3 (desimal)
---------- x (signed)
$0006 = 6 (desimal) ;

Register 17 ß $00, Register 16 ß $06

4. Isi register 16 dikeluarkan pada port C.
5. Isi register 17 dikeluarkan pada port D.

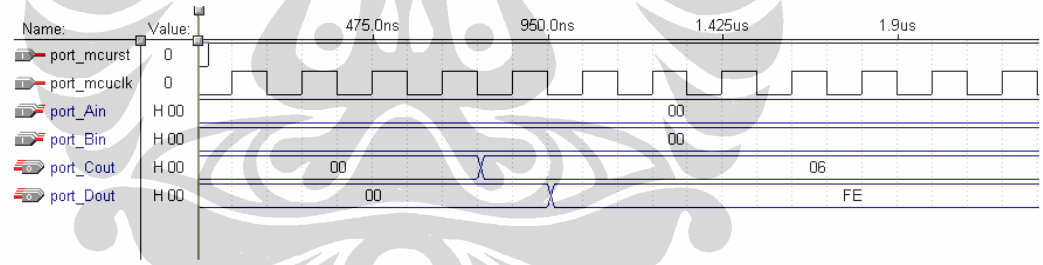
16. PENGUJIAN INSTRUKSI MULSU

Instruksi MULSU (*Multiply Signed with Unsigned*) mempunyai bentuk *MULSU Rd,Rr*. Instruksi ini akan melakukan operasi aritmatik perkalian antara isi register Rd dengan isi register Rr (*R1:R0 ß RdxRr*). Operasi ini melakukan perkalian antara *multiplicant* 8-*bit* bertanda dengan *multiplier* 8-*bit* tak bertanda dan menghasilkan 16-*bit* hasil operasi. Hasil operasi ini akan disimpan pada register 16 untuk *low byte* dan register 17 untuk *high byte*.

Pengujian instruksi MULSU dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi MULSU
ldi r20,$FE     ; register 20 diisi data $FE
ldi r21,$FD     ; register 21 diisi data $FD
mulsu r20,r21   ; operasi MULSU antara isi register 20 dan isi register 21
                ; hasil disimpan pada register 16 dan register 17
out $15,r16     ; isi register 16 dikeluarkan pada port C
out $12,r17     ; isi register 17 dikeluarkan pada port D
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi MULSU adalah sebagai berikut:

1. Register 20 diisi dengan data $FE atau 1111 1110 (biner) atau -2 (desimal).
2. Register 21 diisi dengan data $FD atau 1111 1101 (biner) atau 253 (desimal).
3. Isi register 20 dan isi register 21 dilakukan operasi perkalian dan hasil disimpan pada register 16 (*low byte*) dan register 17 (*high byte*).

Register 20 à $FE = -2 (desimal)
Register 21 à $FD = 253 (desimal)
---------- x (signed)
$FE06 = -506 (desimal) ;

Register 17 ß $FE, Register 16 ß $06

4. Isi register 16 dikeluarkan pada port C.
5. Isi register 17 dikeluarkan pada port D.

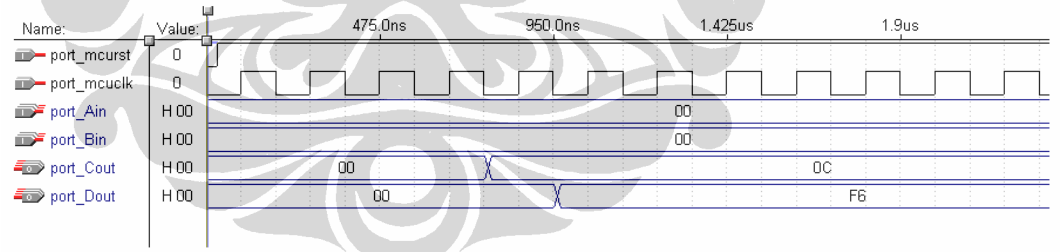
17. PENGUJIAN INSTRUKSI FMUL

Instruksi FMUL (*Fractional Multiply Unsigned*) mempunyai bentuk *FMUL Rd,Rr*. Instruksi ini akan melakukan operasi aritmatik perkalian tak bertanda antara isi register Rd dengan isi register Rr dan hasil digeser 1 *bit* ke kiri (*R1:R0 ß (RdxRr)<<1*). Operasi ini melakukan perkalian antara *multiplicant* 8-*bit* tak bertanda dengan *multiplier* 8-*bit* tak bertanda dan menghasilkan 16-*bit* hasil operasi. Hasil operasi ini akan disimpan pada register 16 untuk *low byte* dan register 17 untuk *high byte.*

Pengujian instruksi FMUL dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi FMUL
ldi r20,$FE     ; register 20 diisi data $FE
ldi r21,$FD     ; register 21 diisi data $FD
fmul r20,r21    ; operasi FMUL antara isi register 20 dan isi register 21
                ; hasil disimpan pada register 16 dan register 17
out $15,r16     ; isi register 16 dikeluarkan pada port C
out $12,r17     ; isi register 17 dikeluarkan pada port D
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi FMUL adalah sebagai berikut:

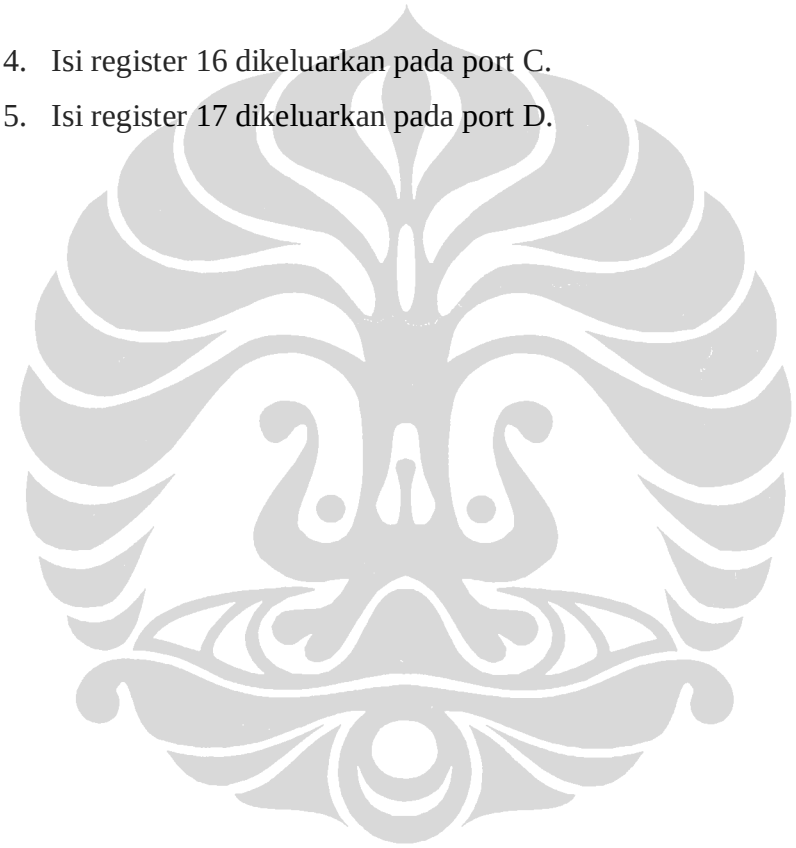
1. Register 20 diisi dengan data $FE atau 1111 1110 (biner) atau 254 (desimal).
2. Register 21 diisi dengan data $FD atau 1111 1101 (biner) atau 253 (desimal).
3. Isi register 20 dan isi register 21 dilakukan operasi perkalian dan hasil disimpan pada register 16 (*low byte*) dan register 17 (*high byte*).

Register 20 à $FE = 254 (desimal)
Register 21 à $FD = 253 (desimal)
---------- x (unsigned)
$FB06 = 64262 (desimal) ;

Hasil $FB06 (B 1111 1011 0000 0110) digeser 1 *bit* ke kiri sehingga menjadi $F60C (B 1111 0110 0000 1100)

Register 17 ß $F6, Register 16 ß $0C

4. Isi register 16 dikeluarkan pada port C.
5. Isi register 17 dikeluarkan pada port D.



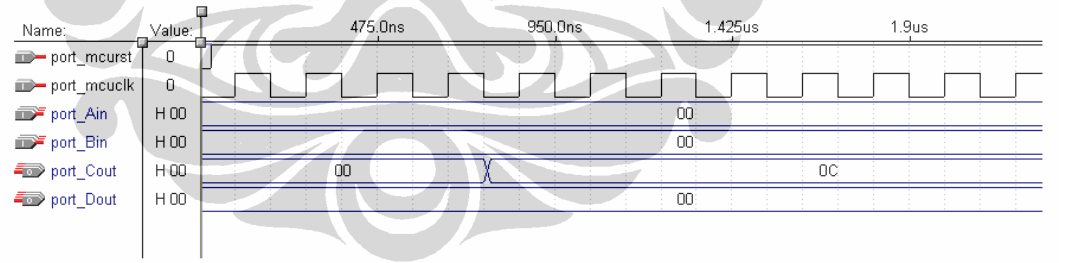
18. PENGUJIAN INSTRUKSI FMULS

Instruksi FMULS (*Fractional Multiplu Signed*) mempunyai bentuk *FMULS Rd,Rr*. Instruksi ini akan melakukan operasi aritmatik perkalian bertanda antara isi register Rd dengan isi register Rr dan hasil digeser 1 *bit* ke kiri (*R1:R0 ß (RdxRr)<<1*). Operasi ini melakukan perkalian antara *multiplicant* 8-*bit* bertanda dengan *multiplier* 8-*bit* bertanda dan menghasilkan 16-*bit* hasil operasi. Hasil operasi ini akan disimpan pada register 16 untuk *low byte* dan register 17 untuk *high byte*.

Pengujian instruksi FMULS dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi FMULS
ldi r20,$FE     ; register 20 diisi data $FE
ldi r21,$FD     ; register 21 diisi data $FD
fmuls r20,r21   ; operasi FMULS antara isi register 20 dan isi register 21
                ; hasil disimpan pada register 16 dan register 17
out $15,r16     ; isi register 16 dikeluarkan pada port C
out $12,r17     ; isi register 17 dikeluarkan pada port D
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi FMULS adalah sebagai berikut:

1. Register 20 diisi dengan data $FE atau 1111 1110 (biner) atau -2 (desimal).
2. Register 21 diisi dengan data $FD atau 1111 1101 (biner) atau -3 (desimal).
3. Isi register 20 dan isi register 21 dilakukan operasi perkalian dan hasil disimpan pada register 16 (*low byte*) dan register 17 (*high byte*).

Register 20 à $FE = -2 (desimal)
Register 21 à $FD = -3 (desimal)
---------- x (unsigned)
$0006 = 6 (desimal) ;

Hasil $0006 (B 0000 0000 0000 0110) digeser 1 *bit* ke kiri sehingga menjadi $000C (B 0000 0000 0000 1100)

Register 17 ß $00, Register 16 ß $0C

4. Isi register 16 dikeluarkan pada port C.
5. Isi register 17 dikeluarkan pada port D.

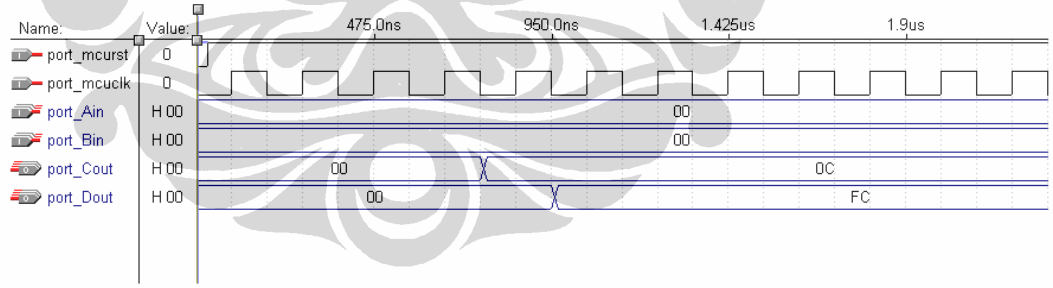
19. PENGUJIAN INSTRUKSI FMULSU

Instruksi FMULSU (*Fractional Multiply Signed with Unsigned*) mempunyai bentuk *FMULSU Rd,Rr*. Instruksi ini akan melakukan operasi aritmatik perkalian bertanda antara isi register Rd dengan isi register Rr dan hasil digeser 1 *bit* ke kiri (*R1:R0 ß (RdxRr)<<1*). Operasi ini melakukan perkalian antara *multiplicant* 8-*bit* bertanda dengan *multiplier* 8-*bit* tak bertanda dan menghasilkan 16-*bit* hasil operasi. Hasil operasi ini akan disimpan pada register 16 untuk *low byte* dan register 17 untuk *high byte.*

Pengujian instruksi FMULSU dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi FMULSU
ldi r20,$FE      ; register 20 diisi data $FE
ldi r21,$FD      ; register 21 diisi data $FD
fmulsu r20,r21; operasi FMULSU antara isi register 20 dan isi register 21
               ; hasil disimpan pada register 16 dan register 17
out $15,r16      ; isi register 16 dikeluarkan pada port C
out $12,r17      ; isi register 17 dikeluarkan pada port D
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi FMULSU adalah sebagai berikut:

1. Register 20 diisi dengan data $FE atau 1111 1110 (biner) atau -2 (desimal).
2. Register 21 diisi dengan data $FD atau 1111 1101 (biner) atau 253 (desimal).
3. Isi register 20 dan isi register 21 dilakukan operasi perkalian dan hasil disimpan pada register 16 (*low byte*) dan register 17 (*high byte*).

Register 20 à $FE = -2 (desimal bertanda)
Register 21 à $FD = 253 (desimal tak bertanda)
---------- x (signed)
$FE06 = -506 (desimal) ;

Hasil $FE06 (B 1111 1110 0000 0110) digeser 1 *bit* ke kiri sehingga menjadi $FC0C (B 1111 1100 0000 1100)

Register 17 ß $FC, Register 16 ß $0C

4. Isi register 16 dikeluarkan pada port C.
5. Isi register 17 dikeluarkan pada port D.

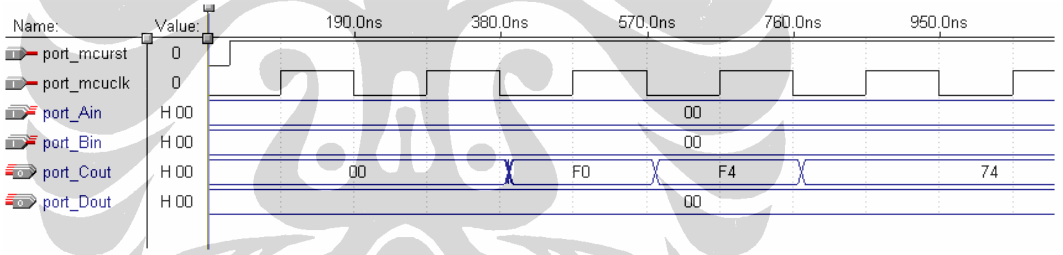
20. PENGUJIAN INSTRUKSI SBI DAN CBI

Instruksi SBI (*Set Bit in I/O Register*) mempunyai bentuk *SBI P,b*. Instruksi ini akan melakukan operasi *bit* pada register I/O. Instruksi ini akan menset *bit* ke-b pada alamat port P (*I/O(P,b)* ß *1*). Instruksi CBI (*Clear Bit in I/O Register*) mempunyai bentuk *CBI P,b*. Instruksi ini akan memberikan logika *low* pada *bit* ke-b pada alamat port P (*I/O(P,b)* ß *0*).

Pengujian instruksi SBI dan CBI dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Pengujian instruksi SBI dan CBI
ldi r16,$F0      ; Register 16 diisi dengan data $F0
out $15,r16      ; Isi register 16 dikeluarkan ke port C
sbi $15,2        ; Set bit ke 2 pada port C
cbi $15,7        ; Clear bit ke 7 pada port C
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi SBI dan CBI adalah sebagai berikut:

1. Register 16 diisi dengan data $F0.
2. Isi register 16 dikeluarkan ke port C sehingga keluarannya $F0 atau B 1111 0000.
3. Set bit ke-2 pada I/O register dengan alamat $15, yaitu port C. Dengan demikian keluarannya menjadi B 1111 0100 atau $F4.
4. Clear bit ke-7 pada I/O register dengan alamat $15, yaitu port C. Dengan demikian keluarannya menjadi B 0111 0100 atau $74.

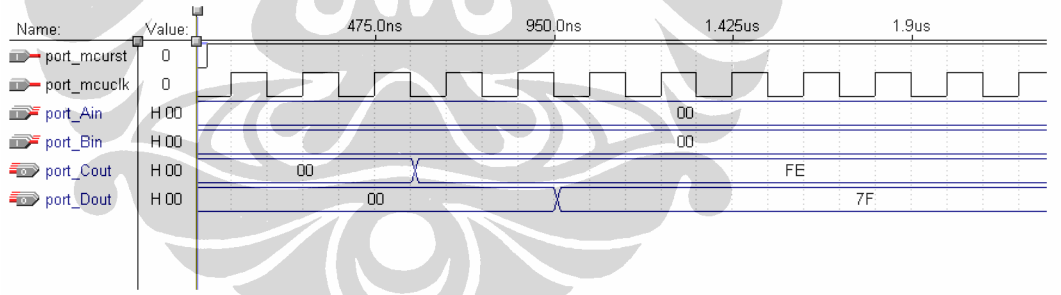
21. PENGUJIAN INSTRUKSI LSL DAN LSR

Instruksi LSL (*Logical Shift Left*) mempunyai bentuk *LSL Rd.* Instruksi ini akan melakukan operasi pergeseran *bit* ke ke kiri dimana *bit* ke-7 akan masuk ke *flag carry* di *status register* (*Rd(n+1)* ß *Rd(n),Rd(0)* ß *0*). Instruksi LSR (*Logical Shift Right*) mempunyai bentuk *LSR Rd* akan melakukan operasi pergeseran *bit* ke ke kanan dimana *bit* ke-0 akan masuk ke *flag carry* di *status register* (*Rd(n)* ß *Rd(n+1),Rd(7)* ß *0*). Kedua operasi ini akan menyimpan kembali hasil operasi pada register Rd.

Pengujian instruksi LSL dan LSR dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi LSL dan LSR
ldi r16,$FF     ; Register 16 diisi dengan data $FF
lsl r16         ; Operasi pergeseran bit ke kiri isi register 16
out $15,r16     ; Isi register 16 dikeluarkan pada port C
lsr r16         ; Operasi pergeseran bit ke kanan isi register 16
out $12,r16     ; Isi register 16 dikeluarkan pada port D
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi LSL dan LSR adalah sebagai berikut:

1. Register 16 diisi dengan data $FF atau B 1111 1111.
2. Operasi pergeseran *bit* ke kiri isi register 16 dimana *bit* ke-7 akan disimpan ke *carry*, sedangkan *bit* ke-0 diisi dengan logika ’0’. Hasil operasi adalah $FE dan *carry* menjadi ‘1’ dan dikeluarkan pada port C. Hasil juga disimpan kembali pada register 16.

   B 1111 1111 ($FF) à B 1111 1110 ($FE) ; carry ß ’1’

3. Operasi pergeseran *bit* ke kanan isi register 16 *bit* ke-7 diisi dengan logika ’0’ sedangkan *bit* ke-0 disimpan ke *carry*. Hasil operasi adalah $7F dan *carry* menjadi ’0’. Hasil dikeluarkan pada port D. Hasil juga disimpan kembali pada register 16.

   B 1111 1110 ($FE) **à** B 0111 1111 ($7F) ; carry **ß** ’0’

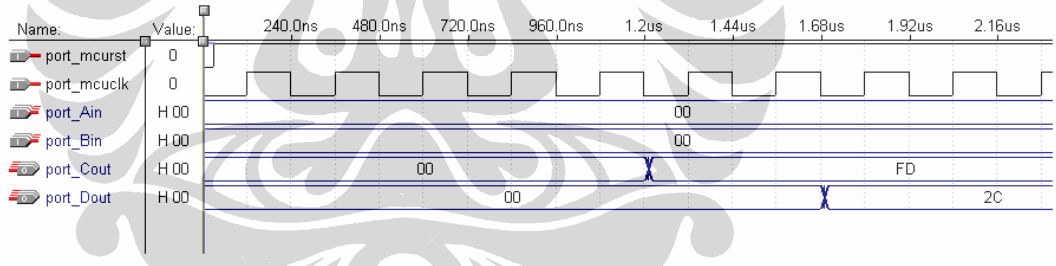
22. PENGUJIAN INSTRUKSI ROL

Instruksi ROL (*Rotate Left Through Carry*) mempunyai bentuk *ROL Rd*. Instruksi ini akan melakukan pergeseran *bit* 1 posisi ke kiri, dimana nilai *carry* yang ada di *status register* akan masuk ke *bit* 0, dan *bit* ke-7 akan kembali mengisi *carry* yang berada di *status register* (*Rd(0)* ß *Carry, Rd(n+1)* ß *Rd(n), Carry* ß *Rd(7)*). Hasil operasi kembali disimpan di register Rd.

Pengujian instruksi ROL dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi ROL
ldi r16,$7E      ; Register 16 diisi data $7E
ldi r18,$01      ; Register 18 diisi data $01
out $3F,r18      ; Set carry
rol r16          ; Operasi "Rotate Left through Carry"
out $15,r16      ; Isi register 16 dikeluarkan pada port C
in r17,$3F       ; Ambil nilai status register dan simpan ke register 17
out $12,r17      ; Isi register 17 dikeluarkan pada port D
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika '0' pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi ROL adalah sebagai berikut:

1. Register 16 diisi dengan data $7E atau B 0111 1110.
2. Register 18 diisi dengan data $01 atau B 0000 0001.
3. $3F adalah alamat dari *status register*. Isi dari register 18 (B 0000 0001) kemudian disimpan pada *status register*. Hal ini menyebabkan nilai *flag carry* (posisi *bit* ke-0) bernilai '1'.

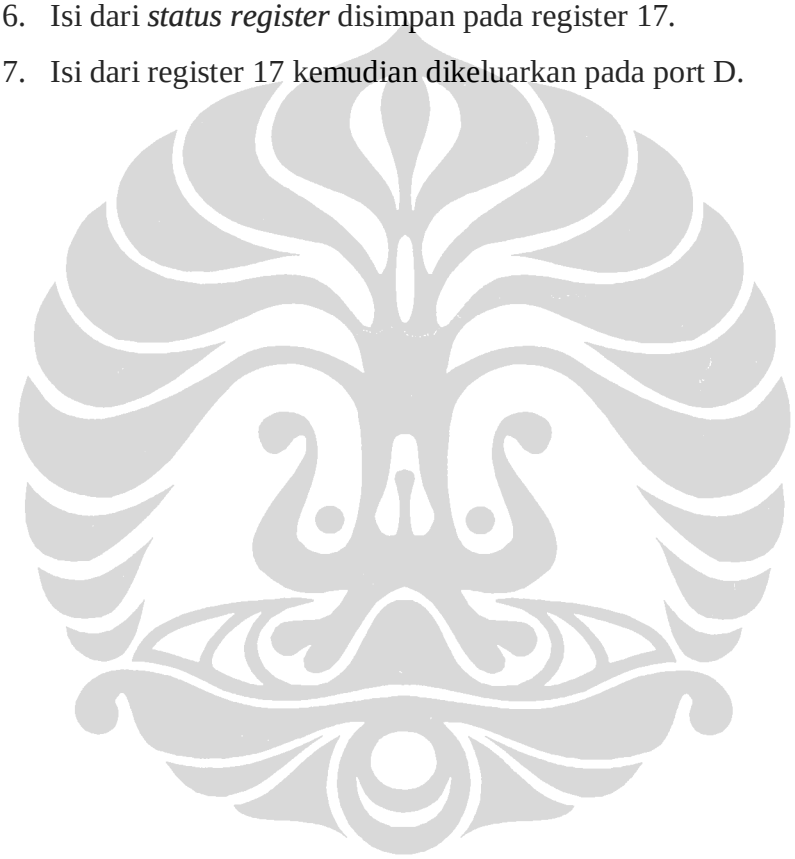
4. Operasi ROL dilakukan terhadap isi register 16 dan hasil operasi disimpan kembali pada register 16.

   $7E **à** B 0111 1110 , *carry* = ’1’;

   Setelah operasi ROL (pergeseran *bit* 1 posisi ke kiri, *carry* masuk ke *bit* 0 dan *bit* 7 masuk ke *carry*) menjadi:

   $FD **à** 1111 1101 , *carry* = ‘0’;

5. Isi dari register 16 yaitu $FD dikeluarkan pada port C.
6. Isi dari *status register* disimpan pada register 17.
7. Isi dari register 17 kemudian dikeluarkan pada port D.



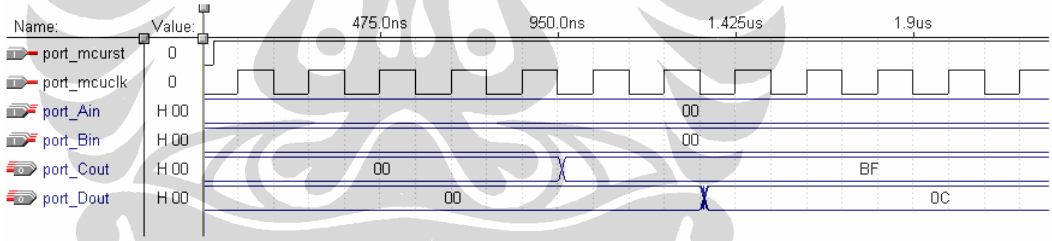
23. PENGUJIAN INSTRUKSI ROR

Instruksi ROR (*Rotate Right Through Carry*) mempunyai bentuk *ROR Rd*. Instruksi ini akan melakukan pergeseran *bit* 1 posisi ke kanan, dimana nilai *carry* yang ada di *status register* akan masuk ke *bit* 7, dan *bit* ke-0 akan kembali mengisi *carry* yang berada di *status register* (*Rd(7)* ß *Carry, Rd(n)* ß *Rd(n+1), Carry* ß *Rd(0)*). Hasil operasi kembali disimpan di register Rd.

Pengujian instruksi ROR dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi ROR
ldi r16,$7E      ; Register 16 diisi data $7E
ldi r18,$01      ; Register 18 diisi data $01
out $3F,r18      ; Set carry
ror r16          ; Operasi Rotate Right through Carry
out $15,r16      ; Isi register 16 dikeluarkan pada port C
in r17,$3F       ; Ambil nilai status register dan simpan ke register 17
out $12,r17      ; Isi register 17 dikeluarkan pada port D
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi ROR adalah sebagai berikut:

1. Register 16 diisi dengan data $7E atau B 0111 1110.
2. Register 18 diisi dengan data $01 atau B 0000 0001.
3. $3F adalah alamat dari *status register*. Isi dari register 18 (B 0000 0001) kemudian disimpan pada status register. Hal ini menyebabkan nilai *flag carry* (posisi *bit* ke-0) bernilai ’1’.

4. Operasi ROR dilakukan terhadap isi register 16 dan hasil operasi disimpan kembali pada register 16.

   $7E **à** B 0111 1110 , *carry* = ’1’;

   Setelah operasi ROR (pergeseran *bit* 1 posisi ke kanan, *carry* masuk ke *bit* 7 dan *bit* 0 masuk ke *carry*) menjadi:

   $BF **à** B 1011 1111 , *carry* = ‘0’;

5. Isi dari register 16 yaitu $BF dikeluarkan pada port C.
6. Isi dari *status register* disimpan pada register 17.
7. Isi dari register 17 kemudian dikeluarkan pada port D.

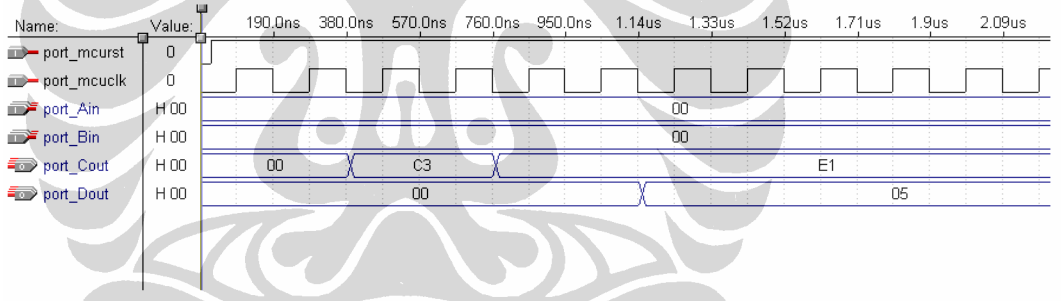
24. PENGUJIAAN INSTRUKSI ASR

Instruksi ASR (*Arithmetic Shift Right*) mempunyai bentuk *ASR, Rd*. Instruksi ini akan melakukan pergeseran *bit* ke kanan, *bit* 0 akan masuk ke *carry*, namun *bit* 7 dibiarkan tetap (*R(n)* ß *Rd(n+1), n=0..6*). Hasil operasi akan kembali disimpan pada register Rd.

Pengujian instruksi ASR dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi ASR
ldi r16,$C3     ; Register 16 diisi dengan data $C3
out $15,r16     ; Isi register 16 dikeluarkan pada port C
asr r16         ; Operasi ASR dilakukan, hasil disimpan kembali pada register 16
out $15,r16     ; Isi register 16 dikeluarkan pada port C
in r17,$3F      ; Nilai status register disimpan pada register 17
out $12,r17     ; Isi register 17 dikeluarkan pada port D
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi ASR adalah sebagai berikut:

1. Register 16 diisi dengan data $C3 atau B 1100 0011.
2. Isi register 16 dikeluarkan pada port C.
3. Operasi ASR dilakukan terhadap isi register 16 dan hasil operasi kembali disimpan pada register 16.

   $C3 à B 1100 0011 ; *carry*=’0’

Setelah operasi ASR (pergeseran *bit* 1 posisi ke kanan, *bit* 0 masuk *carry*, bit 7 tetap) menjadi:

$E1 **à** B 1110 0001 ; *carry*='1'

4. Isi dari register 16 yaitu $E1 dikeluarkan pada port C.
5. Isi dari *status register* disimpan pada register 17.
6. Isi dari register 17 kemudian dikeluarkan pada port D.

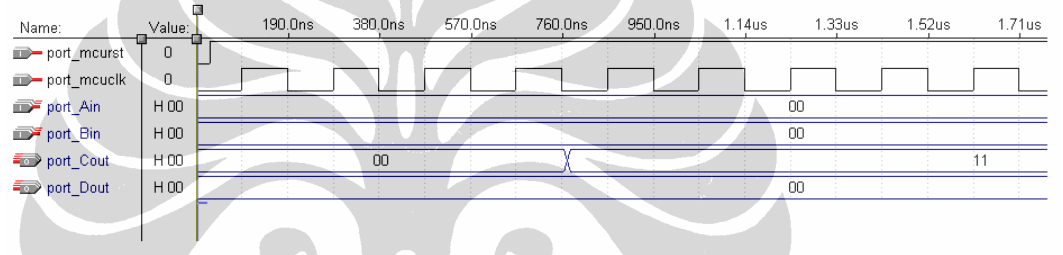
25. PENGUJIAN INSTRUKSI BSET

Instruksi BSET (*Flag Set*) mempunyai bentuk *BSET s.* Instruksi ini akan memberikan nilai logika ’1’ pada *status register* posisi *bit* ke-s (*SREG(s)* ß *1*).

Pengujian instruksi BSET dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi BSET
bset 0          ; Set posisi bit 0 pada status register
bset 4          ; Set posisi bit 4 pada status register
in r16,$3F      ; Ambil nilai status register dan simpan pada register 16
out $15,r16     ; Isi register 16 dikeluarkan pada port C
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi BSET adalah sebagai berikut:

1. Set posisi *bit* ke-0 pada *status register*.
2. Set posisi *bit* ke-4 pada *status register*.
3. Isi dari *status* register, yaitu $11 atau B 0001 0001 disimpan pada register 16.
4. Isi dari register 16 dikeluarkan pada port C.

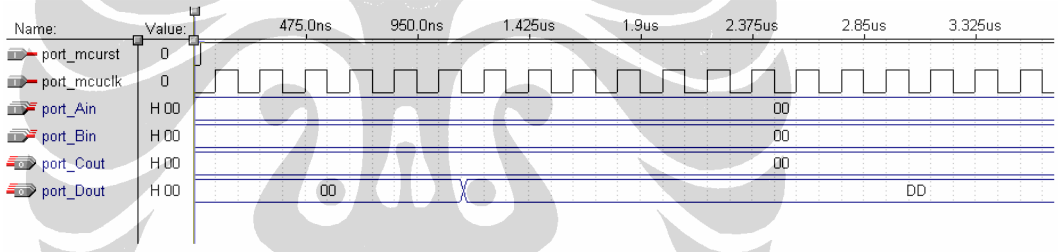
26. PENGUJIAN INSTRUKSI BCLR

Instruksi BCLR (*Flag Clear*) mempunyai bentuk *BCLR s*. Instruksi ini akan memberikan nilai logika ’0’ pada *status register* posisi *bit* ke-s (*SREG(s)* ß *0*).

Pengujian instruksi BCLR dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi BCLR
ldi r16,$FF     ; Register 16 diisi data $FF
out $3F,r16     ; Isi register 16 disimpan ke status register
bclr 1          ; Clear posisi bit 1 pada status register
bclr 5          ; Clear posisi bit 5 pada status register
in r17,$3F      ; Ambil nilai status register dan simpan pada register 17
out $12,r17     ; Isi register 17 dikeluarkan pada port D
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi BCLR adalah sebagai berikut:

1. Register 16 diisi dengan data $FF atau B 1111 1111.
2. Isi register 16, yaitu $FF, disimpan ke *status register* sehingga isi *status register* menjadi B 1111 1111.
3. *Clear* posisi *bit* 1 di *status register*.
4. *Clear* posisi *bit* 5 di *status register*.
5. Isi dari *status register*, yaitu B 1101 1101 atau $DD,disimpan pada register 17.
6. Isi dari register 17 dikeluarkan pada port D.

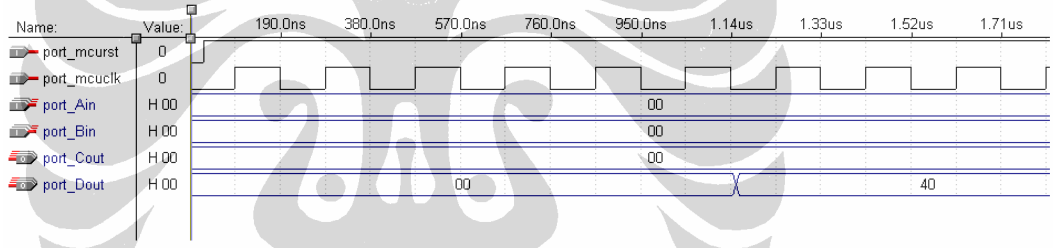
27. PENGUJIAN INSTRUKSI BST

Instruksi BST (*Bit Store from Register to T*) mempunyai bentuk *BST Rr,b*. Instruksi ini akan melakukan penyimpanan *bit* posisi ke-b dari register Rr ke *flag T* di *status register* (*T ß Rr(b)*).

Pengujian instruksi BST dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi BST
in r17,$3F      ; isi status register disimpan ke register 17
out $12,r17     ; isi register 17 dikeluarkan pada port D
ldi r16,$02     ; register 16 diisi data $02 atau B0000 0010
bst r16,1       ; ambil bit posisi 1 dari register 16 dan simpan ke flag T
in r17,$3F      ; isi status register disimpan ke register 17
out $12,r17     ; isi register 17 dikeluarkan pada port D
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi BST adalah sebagai berikut:

1. Isi dari *status register* dengan alamat $3F disimpan pada register 17.
2. Isi dari register 17 ini dikeluarkan pada port D.
3. Register 16 diisi data $02 atau B 0000 0010.
4. Perintah *bst r16,1* akan mengambil isi register 16 posisi *bit* 1 dan disimpan ke *flag* T di *status register*.
5. Isi dari *status register* dengan alamat $3F, yaitu $40 atau B 0100 0000 (*flag* T posisi *bit* 6), diambil kembali ke register 17.
6. Isi dari register 17 ini dikeluarkan kembali ke port D.

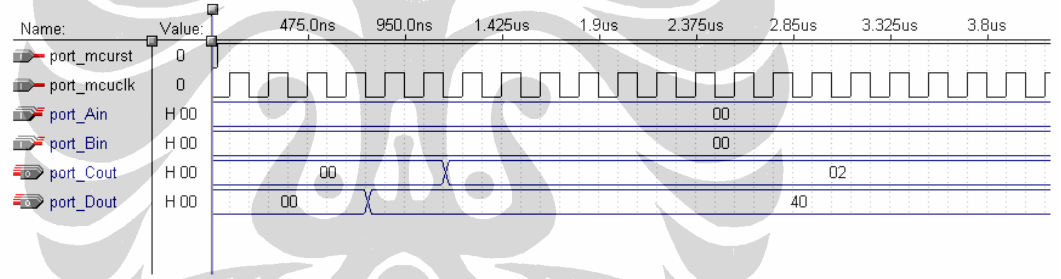
28. PENGUJIAN INSTRUKSI BLD

Instruksi BLD (*Bit load from T to Register*) mempunyai bentuk *BLD Rd,b*. Instruksi ini akan melakukan penyimpanan *bit* dari *flag T* di *status register* ke register Rd pada posisi *bit* ke-b (*Rd(b)* ß *T*).

Pengujian instruksi BLD dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi BLD
ldi r18,$40      ; register 18 diisi data $40 atau B 0100 0000
out $3F,r18      ; isi register 18 disimpan ke status register
in r17,$3F       ; isi status register disimpan ke register 17
out $12,r17      ; isi register 17 dikeluarkan ke port D
bld r16,$01      ; isi flag T status register disimpan ke register 16 posisi bit 1
out $15,r16      ; isi register 16 dikeluarkan ke port C
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi BLD adalah sebagai berikut:

1. Register 18 diisi data $40 atau B 0100 0000.
2. Isi dari register 18 disimpan ke *status register*, dengan demikian *status register* akan berisi B 0100 0000 atau *flag* T akan bernilai 1.
3. Isi dari *status register* kemudian disimpan kembali ke register 17.
4. Isi dari register 17 dikeluarkan pada port D dengan alamat $12.
5. Operasi BLD akan menyimpan isi *flag* T, yaitu 1 ke register 16 dengan posisi bit 1, sehingga register 16 akan berisi B 0000 0010 atau $02.
6. Isi dari register 16 dikeluarkan ke port C dengan alamat $15.

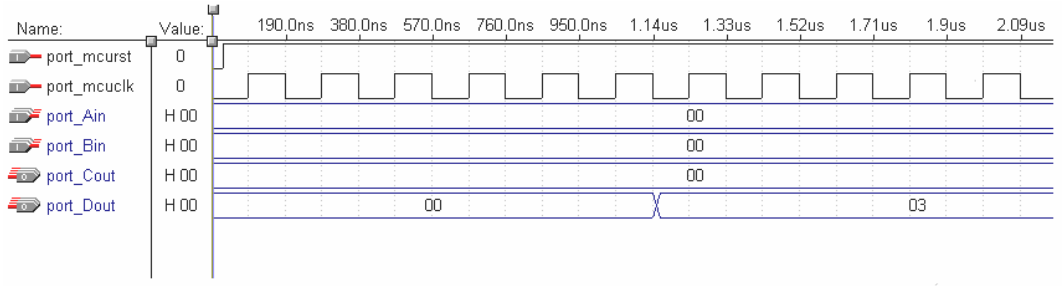
29. PENGUJIAN INSTRUKSI SEC, SEN, SEZ, SEI, SES, SEV, SET, SEH

Kelompok instruksi ini akan memberikan logika ’1’ pada *flag* di *status register* yang bersangkutan. Instruksi SEC (*Set Carry*) akan memberikan logika ’1’ pada *flag carry* (*C*ß*1*), instruksi SEN (*Set Negative Flag*) akan memberikan logika ’1’ pada *flag negative* (*N*ß*1*), instruksi SEZ (*Set Zero Flag*) akan memberikan logika ’1’ pada *flag zero* (*Z*ß*1*), instruksi SEI (*Global Interrupt Enable*) akan memberikan logika ’1’ pada *flag global interrupt* (*I*ß*1*), instruksi SES (*Set Signed Test Flag*) akan memberikan logika ’1’ pada *flag signed test* (*S*ß*1*), instruksi SEV (*Set Two’s Complement Overflow*) akan memberikan logika ’1’ pada *flag twos complement overflow* (*V*ß*1*), instruksi SET (*Set T in SREG*) akan memberikan logika ’1’ pada *flag T* (*T*ß*1*), dan instruksi SEH (*Set Half Carry Flag in SREG*) akan memberikan logika ’1’ pada *flag half carry* (*H*ß*1*).

Pengujian kelompok instruksi ini dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya: Pengujian dilakukan bertahap, yaitu pengujian pertama untuk instruksi SEC dan SEZ, pengujian kedua untuk instruksi SEN dan SEV, pengujian ketiga untuk instruksi SES dan SEH, pengujian terakhir untuk instruksi SET dan SEI.

```
; Pengujian instruksi SEC dan SEZ
in r16,$3F      ; isi status register disimpan ke register 16
out $15,r16     ; isi register 16 dikeluarkan pada port C
sec             ; operasi set bit 0 di status register  untuk flag C
sez             ; operasi set bit 1 di status register  untuk flag Z
in r16,$3F      ; isi status register disimpan ke register 16
out $12,r16     ; isi register 16 dikeluarkan pada port D
```

Hasil simulasi:

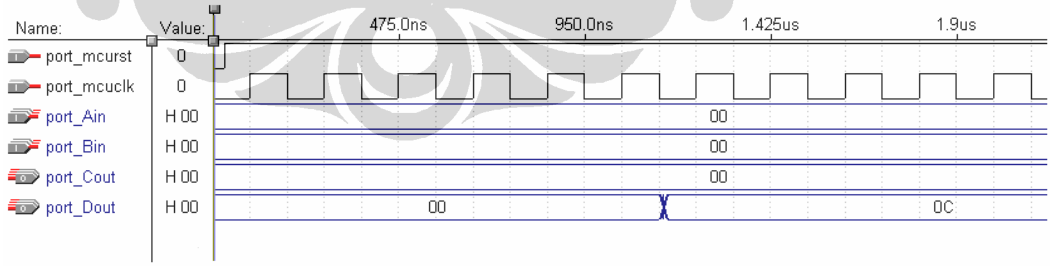


Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi SEC dan SEZ adalah sebagai berikut:

1. Isi dari *status register* disimpan pada register 16.
2. Isi dari register 16 dikeluarkan pada port C, hal ini untuk mengecek isi *status register*.
3. Instruksi SEC akan memberikan logika ’1’ terhadap *flag* C pada *status register* (posisi bit 0).
4. Instruksi SEZ akan memberikan logika ’1’ terhadap *flag* Z pada *status register* (posisi bit 1).
5. Isi dari *status register* kembali disimpan ke register 16.
6. Isi dari register 16 dikeluarkan pada port D, yaitu B 0000 0011 atau $03.

```
; Pengujian instruksi SEN dan SEV
in r16,$3F      ; isi status register disimpan ke register 16
out $15,r16     ; isi register 16 dikeluarkan pada port C
sen             ; operasi set bit 2 di status register  untuk flag N
sev             ; operasi set bit 3 di status register  untuk flag V
in r16,$3F      ; isi status register disimpan ke register 16
out $12,r16     ; isi register 16 dikeluarkan pada port D
```

Hasil simulasi:



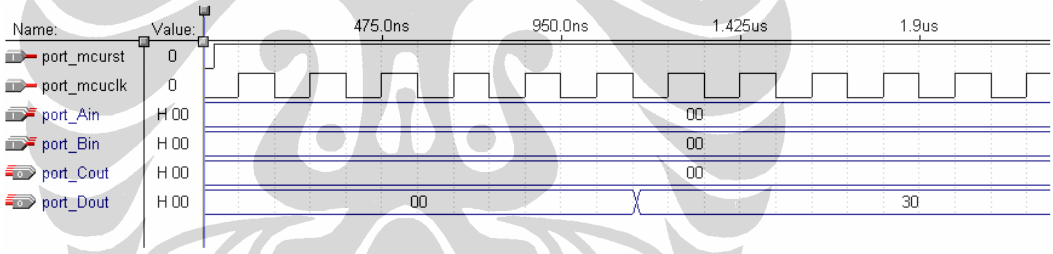
Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi SEN dan SEV adalah sebagai berikut:

1. Isi dari *status register* disimpan pada register 16.
2. Isi dari register 16 dikeluarkan pada port C, hal ini untuk mengecek isi *status register*.

3. Instruksi SEC akan memberikan logika ’1’ terhadap *flag* N pada *status register* (posisi bit 2).
4. Instruksi SEZ akan memberikan logika ’1’ terhadap *flag* V pada *status register* (posisi bit 3).
5. Isi dari *status register* kembali disimpan ke register 16.
6. Isi dari register 16 dikeluarkan pada port D, yaitu B 0000 1100 atau $0C.

```
; Pengujian instruksi SES dan SEH
in r16,$3F      ; isi status register disimpan ke register 16
out $15,r16     ; isi register 16 dikeluarkan pada port C
ses             ; operasi set bit 4 di status register  untuk flag S
seh             ; operasi set bit 5 di status register  untuk flag H
in r16,$3F      ; isi status register disimpan ke register 16
out $12,r16     ; isi register 16 dikeluarkan pada port D
```

Hasil simulasi:

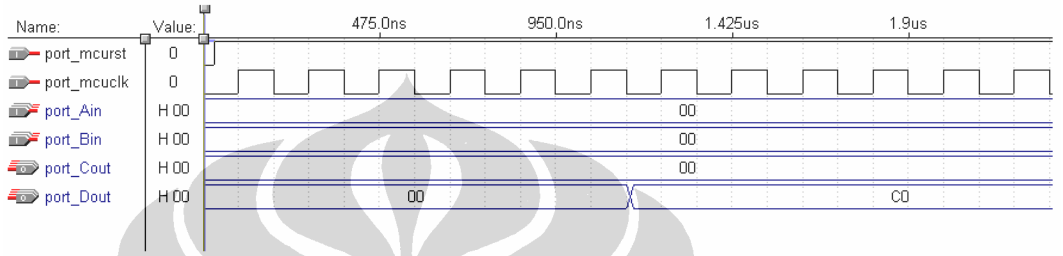


Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi SES dan SEH adalah sebagai berikut:

1. Isi dari *status register* disimpan pada register 16.
2. Isi dari register 16 dikeluarkan pada port C, hal ini untuk mengecek isi *status register*.
3. Instruksi SES akan memberikan logika ’1’ terhadap *flag* S pada *status register* (posisi bit 4).
4. Instruksi SEH akan memberikan logika ’1’ terhadap *flag* H pada *status register* (posisi bit 5).
5. Isi dari *status register* kembali disimpan ke register 16.
6. Isi dari register 16 dikeluarkan pada port D, yaitu B 0011 0000 atau $30.

```
; Pengujian instruksi SET dan SEI
in r16,$3F      ; isi status register disimpan ke register 16
out $15,r16     ; isi register 16 dikeluarkan pada port C
set             ; operasi set bit 6 di status register  untuk flag T
sei             ; operasi set bit 7 di status register  untuk flag I
in r16,$3F      ; isi status register disimpan ke register 16
out $12,r16     ; isi register 16 dikeluarkan pada port D
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi SET dan SEI adalah sebagai berikut:

1. Isi dari *status register* disimpan pada register 16.
2. Isi dari register 16 dikeluarkan pada port C, hal ini untuk mengecek isi *status register*.
3. Instruksi SET akan memberikan logika ’1’ terhadap *flag* T pada *status register* (posisi bit 4).
4. Instruksi SEI akan memberikan logika ’1’ terhadap *flag* I pada *status register* (posisi bit 5).
5. Isi dari *status register* kembali disimpan ke register 16.
6. Isi dari register 16 dikeluarkan pada port D, yaitu B 1100 0000 atau $C0.

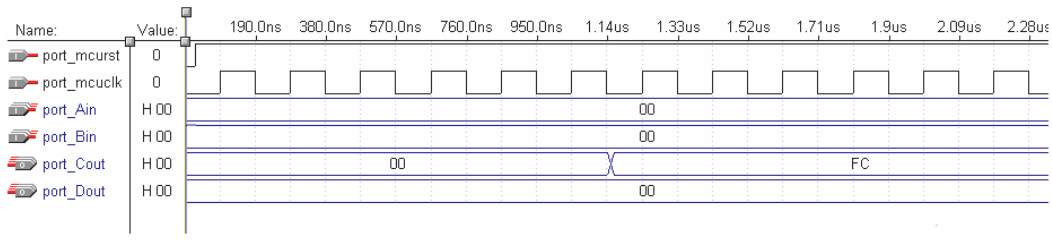
30. PENGUJIAN INSTRUKSI CLC,CLN,CLZ,CLI,CLS,CLV,CLT,CLH

Kelompok instruksi ini akan memberikan logika ’0’ pada *flag* di *status register* yang bersangkutan. Instruksi CLC (*Clear Carry*) akan memberikan logika ’0’ pada *flag carry* (*C*ß*0*), instruksi CLN (*Clear Negative Flag*) akan memberikan logika ’0’ pada *flag negative* (*N*ß*0*), instruksi CLZ (*Clear Zero Flag*) akan memberikan logika ’0’ pada *flag zero* (*Z*ß*0*), instruksi CLI (*Global Interrupt Disable*) akan memberikan logika ’0’ pada *flag global interrupt* (*I*ß*0*), instruksi CLS (*Clear Signed Test Flag*) akan memberikan logika ’0’ pada *flag signed test* (*S*ß*0*), instruksi CLV (*Clear Two’s Complement Overflow*) akan memberikan logika ’0’ pada *flag twos complement overflow* (*V*ß*0*), instruksi CLT (*Clear T in SREG*) akan memberikan logika ’0’ pada *flag T* (*T*ß*0*), dan instruksi CLH (*Clear Half Carry Flag in SREG*) akan memberikan logika ’0’ pada *flag half carry* (*H*ß*0*).

Pengujian kelompok instruksi ini dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya: Pengujian kelompok instruksi ini dilakukan secara bertahap. Pengujian pertama terhadap instruksi CLC dan CLZ, pengujian kedua terhadap instruksi CLN dan CLV, pengujian ketiga terhadap CLS dan CLH, dan pengujian terakhir terhadap CLT dan CLI.

```
; Pengujian instruksi CLC dan CLZ
ldi r16,$FF     ; register 16 diisi data $FF atau B 1111 1111
out $3F,r16     ; isi register 16 disimpan pada status register
clc             ; operasi clear bit pada status register untuk flag C
clz             ; operasi clear bit pada status register untuk flag Z
in r17,$3F      ; isi status register disimpan pada register 17
out $15,r17     ; isi register 17 dikeluarkan pada port C
```

Hasil simulasi:

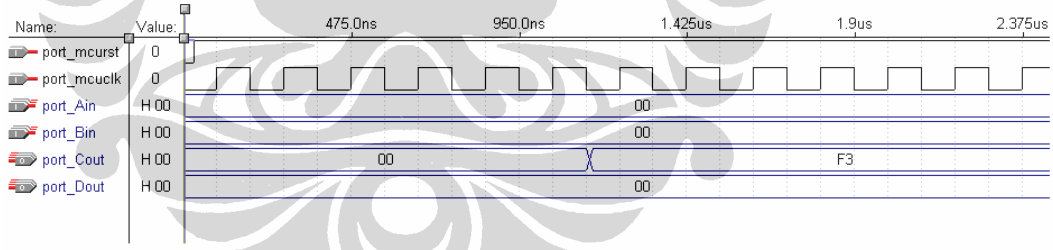


Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi CLC dan CLZ adalah sebagai berikut:

1. Register 16 diisi data $FF atau B 1111 1111.
2. Isi dari register 16 disimpan ke *status register*, sehingga *status register* berisi B 1111 1111.
3. Operasi CLC akan meng-*clear bit flag* C, yaitu posisi *bit* 0 di *status register*.
4. Operasi CLZ akan meng-*clear bit flag* Z, yaitu posisi *bit* 1 di *status register*.
5. Isi dari *status register*, yaitu B 1111 1100 atau $FC, disimpan ke register 17.
6. Isi dari register 17 kemudian dikeluarkan pada port C dengan alamat $15.

```
; Pengujian instruksi CLN dan CLV
ldi r16,$FF     ; register 16 diisi data $FF atau B 1111 1111
out $3F,r16     ; isi register 16 disimpan pada status register
cln             ; operasi clear bit pada status register untuk flag N
clv             ; operasi clear bit pada status register untuk flag V
in r17,$3F      ; isi status register disimpan pada register 17
out $15,r17     ; isi register 17 dikeluarkan pada port C
```

Hasil simulasi:



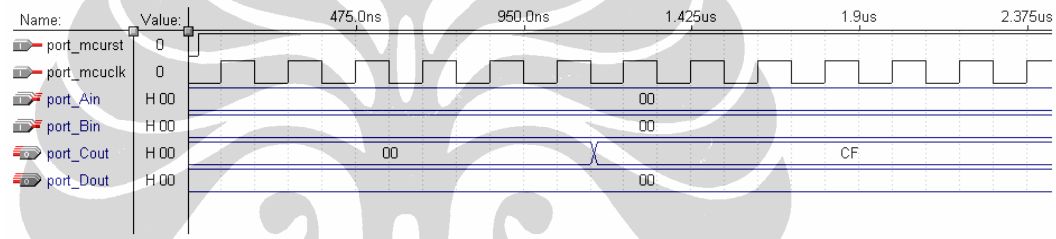
Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi CLN dan CLV adalah sebagai berikut:

1. Register 16 diisi data $FF atau B 1111 1111.
2. Isi dari register 16 disimpan ke *status register*, sehingga *status register* berisi B 1111 1111.
3. Operasi CLN akan meng-*clear bit flag* N, yaitu posisi *bit* 2 pada *status register*.

4. Operasi CLV akan meng-*clear bit flag* V, yaitu posisi *bit* 3 pada *status register*.
5. Isi dari *status register*, yaitu B 1111 0011 atau $F3, disimpan pada register 17.
6. Isi dari register 17 kemudian dikeluarkan pada port C dengan alamat $15.

```
; Pengujian instruksi CLS dan CLH
ldi r16,$FF     ; register 16 diisi data $FF atau B 1111 1111
out $3F,r16     ; isi register 16 disimpan pada status register
cls             ; operasi clear bit pada status register untuk flag S
clh             ; operasi clear bit pada status register untuk flag H
in r17,$3F      ; isi status register disimpan pada register 17
out $15,r17     ; isi register 17 dikeluarkan pada port C
```

Hasil simulasi:

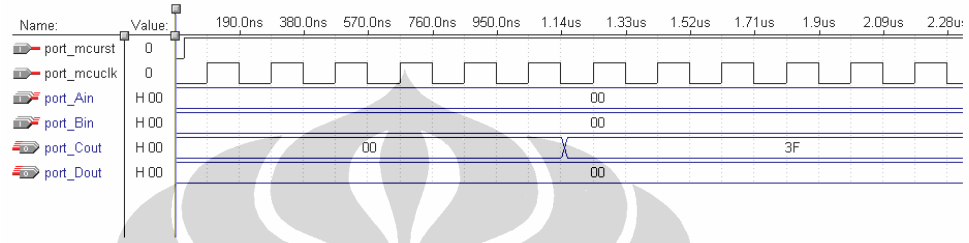


Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi CLS dan CLH adalah sebagai berikut:

1. Register 16 diisi data $FF atau B 1111 1111.
2. Isi dari register 16 disimpan ke *status register*, sehingga *status register* berisi B 1111 1111.
3. Operasi CLS akan meng-*clear bit flag* S, yaitu posisi *bit* 4 di *status register*.
4. Operasi CLH akan meng-*clear bit flag* H, yaitu posisi *bit* 5 di *status register*.
5. Isi dari *status register*, yaitu B 1100 1111 atau $CF, disimpan ke register 17.
6. Isi dari register 17 kemudian dikeluarkan pada port C dengan alamat $15.

```
; Pengujian instruksi CLT dan CLI
ldi r16,$FF     ; register 16 diisi data $FF atau B 1111 1111
out $3F,r16     ; isi register 16 disimpan pada status register
clt             ; operasi clear bit pada status register untuk flag T
cli             ; operasi clear bit pada status register untuk flag I
in r17,$3F      ; isi status register disimpan pada register 17
out $15,r17     ; isi register 17 dikeluarkan pada port C
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika '0' pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi CLT dan CLI adalah sebagai berikut:

1. Register 16 diisi data $FF atau B 1111 1111.
2. Isi dari register 16 disimpan ke *status register*, sehingga *status register* berisi B 1111 1111.
3. Operasi CLT akan meng-*clear bit flag* T, yaitu posisi *bit* 6 di *status register*.
4. Operasi CLI akan meng-*clear bit flag* I, yaitu posisi *bit* 7 di *status register*.
5. Isi dari *status register*, yaitu B 0011 1111 atau $3F, disimpan ke register 17.
6. Isi dari register 17 kemudian dikeluarkan pada port C dengan alamat $15.

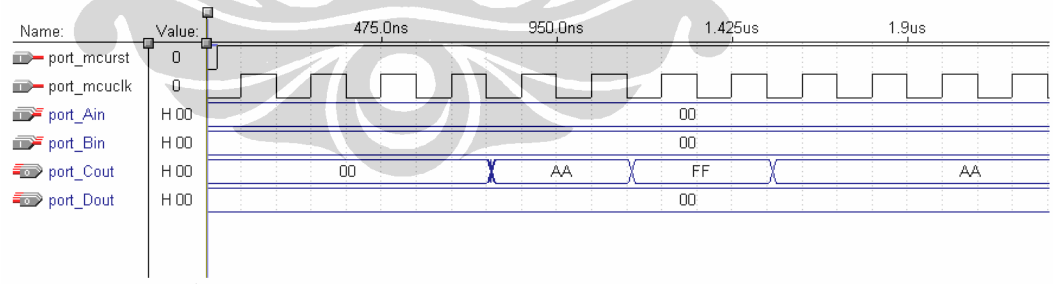
31. PENGUJIAN INSTRUKSI RJMP, RCALL, RET

Instruksi RJMP (*Relative Jump*) mempunyai bentuk *RJMP k.* Instruksi ini akan melakukan percabangan tidak bersyarat ke lokasi *memory* ROM sejauh k (*PC ß PC+k+1*). Instruksi RCALL (*Relative Subroutine Call*) mempunyai bentuk *RCALL k.* Instruksi ini akan melakukan percabangan tidak bersyarat ke rutin ke lokasi *memory* ROM sejauh k (*PC ß PC+k+1*). Alamat sebelum percabangan dilakukan disimpan pada *stack.* Setelah rutin dilakukan maka alamat kembali diambil dari *stack* dengan menggunakan instruksi RET.

Pengujian instruksi RJMP, RCALL, dan RET dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi RCALL, RJMP, dan RET
ldi r16,$AA          ; register 16 diisi data $AA
ldi r17,$FF          ; register 17 diisi data $FF
rcall lompat         ; pemanggilan rutin lompat
out $15,r17          ; isi register 17 dikeluarkan pada port C
rjmp akhir           ; percabangan ke akhir
lompat: out $15,r16  ; isi register 16 dikeluarkan ke port C
ret                  ; kembali ke alamat sebelum pemanggilan rutin
akhir: out $15,r16   ; isi register 16 dikeluarkan ke port C
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi RCALL, RJMP, dan RET adalah sebagai berikut:

7. Register 16 diisi dengan data $AA.
8. Register 17 diisi dengan data $FF.

9. Program kemudian akan memanggil rutin lompat, sehingga eksekusi program dilanjutkan ke alamat rutin lompat. Alamat selanjutnya sebelum pemanggilan rutin dilakukan disimpan ke dalam *stack*.
10. Rutin lompat dijalankan sehingga yang kemudian dilakukan adalah mengeluarkan isi register 16, yaitu $AA, ke port C
11. Instruksi RET akan mengembalikan alamat sebelum pemanggilan rutin dilakukan, sehingga instruksi selanjutnya yang dilakukan adalah *out $15,r17* yang akan mengeluarkan isi register 17, yaitu $FF, ke port C
12. Instruksi RJMP akan melakukan percabangan ke alamat yang ditunjuk oleh label akhir, yaitu merujuk ke instruksi *out $15,r16* yang akan mengeluarkan isi register 16, yaitu $AA, kembali ke port C

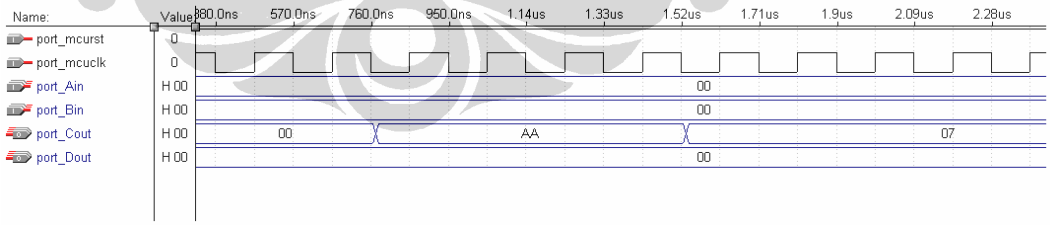
32. PENGUJIAN INSTRUKSI IJMP DAN ICALL

Instruksi IJMP (*Indirect Jump to (Z)*) akan melakukan percabangan tidak bersyarat ke lokasi *memory* ROM pada alamat yang disimpan pada register Z (*PC ß Z*). Instruksi ICALL (*Indirect Call to (Z)*) akan melakukan percabangan tidak bersyarat ke rutin dengan lokasi *memory* ROM di alamat yang disimpan pada register Z (*PC ß Z*). Alamat sebelum percabangan dilakukan disimpan pada *stack*. Setelah rutin dilakukan maka alamat kembali diambil dari *stack* dengan menggunakan instruksi RET.

Pengujian instruksi IJMP, ICALL, dan RET dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi IJMP dan ICALL
ldi r16,$AA          ; register 16 diisi dengan data $AA
ldi r30,$05          ; register 30 atau Z low diisi dengan data $05
icall                ; pemanggilan rutin dengan alamat di register Z
                     ; alamat setelah icall disimpan di stack
ldi r30,$07          ; register 30 atau Z low diisi dengan data $07
ijmp                 ; pemanggilan rutin dengan alamat di register Z
out $15,r16          ; isi register 16 dikeluarkan ke port C
ret                  ; kembali ke alamat yang tersimpan di stack
out $15,r30          ; isi register 30 dikeluarkan ke port C
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi ICALL, IJMP, dan RET adalah sebagai berikut:

1. Register 16 diisi dengan data $AA.
2. Register 30 atau Z *low* diisi dengan data $05.

3. Instruksi ICALL akan memanggil rutin dengan alamat yang tersimpan di register Z, yaitu $05, sehingga eksekusi instruksi selanjutnya adalah *out $15,r16* yang akan mengeluarkan isi register 16, yaitu $AA, ke port C. Alamat instruksi setelah instruksi ICALL disimpan ke dalam *stack*.
4. Instruksi RET akan mengembalikan eksekusi program ke *ldi r30,$07*.
5. Instruksi IJMP akan melakukan percabangan ke alamat yang tersimpan di register Z, yaitu $07, sehingga eksekusi program selanjutnya adalah instruksi *out $15,r30* yang akan mengeluarkan isi register 30, yaitu $07, ke port C

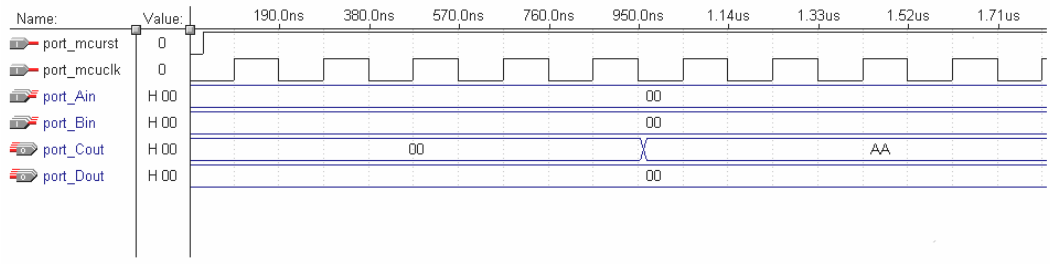
33. PENGUJIAN INSTRUKSI BRBS, BRCS, BRLO, BREQ, BRMI, BRVS, BRLT, BRHS, BRTS, DAN BRIE

Instruksi BRBS (*Branch if Status Flag Set*) mempunyai bentuk *BRBS s,k*. Instruksi ini akan melakukan percabangan bersyarat dengan mengevaluasi *flag* tertentu di *status register* (*if SREG(s)=1 then PC ß PC+k+1*). Percabangan akan dilakukan bila kondisi terpenuhi dan alamat percabangan adalah sejauh *k*. Pengujian terhadap instruksi ini dilakukan sekaligus untuk menguji instruksi *BRCS k* (*Branch if Carry Set*), *BRLO k* (*Branch if Lower*), *BREQ k* (*Branch if Equal*), *BRMI k* (*Branch if Minus*), *BRVS k* (*Branch if Overflow Flag is Set*), *BRLT k* (*Branch if Less Than Zero*), *BRHS k* (*Branch if Half Carry Flag Set*), *BRTS k* (*Branch if T Flag Set*), dan *BRIE k* (*Branch if Interrupt Enable*). Instruksi ***BRCS k*** sama dengan instruksi *BRBS 0,k*. Instruksi *BRLO k* sama dengan instruksi *BRBS 0,k*. Instruksi *BREQ k* sama dengan instruksi *BRBS 1,k*. Instruksi *BRMI k* sama dengan instruksi *BRBS 2,k*. Instruksi *BRVS k* sama dengan instruksi *BRBS 3,k*. Instruksi *BRLT k* sama dengan instruksi *BRBS 4,k*. Instruksi *BRHS k* sama dengan instruksi *BRBS 5,k*. Instruksi *BRTS k* sama dengan instruksi *BRBS 6,k*. Instruksi *BRIE k* sama dengan instruksi *BRBS 7,k*.

Pengujian kelompok instruksi ini dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi BRCS (BRBS 0,k)
ldi r16,$AA          ; isi register 16 dengan data $AA
ldi r18,$FF          ; isi register 18 dengan data $FF
out $3F,r18          ; isi register 18 disimpan ke status register
brcs brcscabang      ; bila flag C=1 maka percabangan dilakukan ke brcscabang
out $15,r18          ; isi register 18 dikeluarkan ke port C
brcscabang: out $15,r16      ; isi register 16 dikeluarkan ke port C
```

Hasil simulasi:

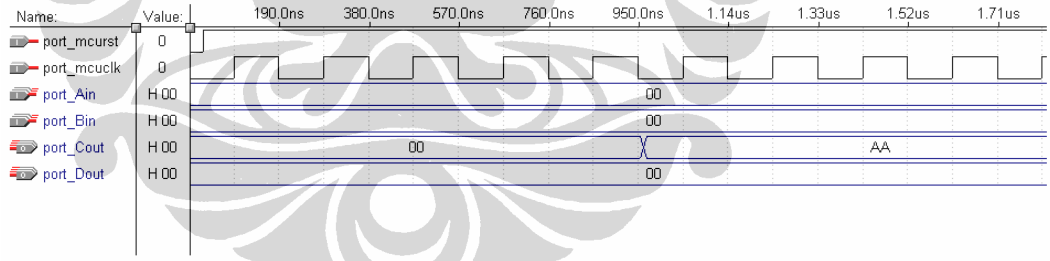


Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi BRCS adalah sebagai berikut:

1. Register 16 diisi dengan data $AA.
2. Register 18 diisi dengan data $FF.
3. *Status register* diisi dengan isi dari register 18, yaitu $FF atau B 1111 1111.
4. Instruksi BRCS akan mengecek bilamana *flag* C bernilai 1 maka percabangan dilakukan.
5. Instruksi selanjutnya yang dieksekusi adalah *out $15,r16* yang akan mengeluarkan isi register 16, yaitu $AA ke port C.

```
; Program uji instruksi BRLO (BRBS 0,k)
ldi r16,$AA             ; isi register 16 dengan data $AA
ldi r18,$FF             ; isi register 18 dengan data $FF
out $3F,r18             ; isi register 18 disimpan ke status register
brlo brlocabang         ; bila flag C=1 maka percabangan dilakukan ke brlocabang
out $15,r18             ; isi register 18 dikeluarkan ke port C
brlocabang: out $15,r16         ; isi register 16 dikeluarkan ke port C
```

Hasil simulasi:



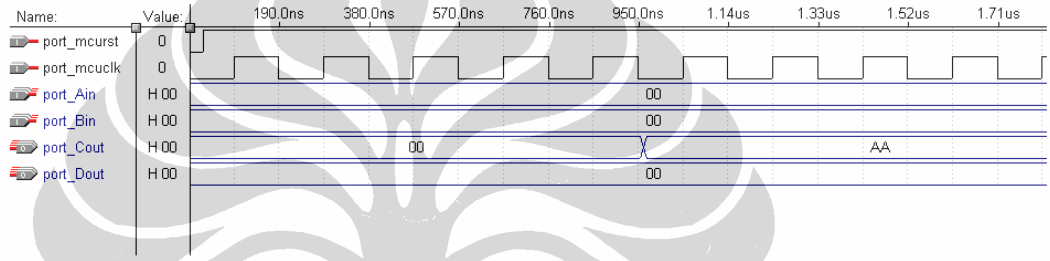
Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi BRLO adalah sebagai berikut:

1. Register 16 diisi dengan data $AA.
2. Register 18 diisi dengan data $FF.
3. *Status register* diisi dengan isi dari register 18, yaitu $FF atau B 1111 1111.
4. Instruksi BRLO akan mengecek bilamana *flag* C bernilai 1 maka percabangan dilakukan.

5. Instruksi selanjutnya yang dieksekusi adalah *out $15,r16* yang akan mengeluarkan isi register 16, yaitu $AA ke port C

```
; Program uji instruksi BREQ (BRBS 1,k)
ldi r16,$AA          ; isi register 16 dengan data $AA
ldi r18,$FF          ; isi register 18 dengan data $FF
out $3F,r18          ; isi register 18 disimpan ke status register
breq breqcabang      ; bila flag Z=1 maka percabangan dilakukan ke breqcabang
out $15,r18          ; isi register 18 dikeluarkan ke port C
brlocabang: out $15,r16      ; isi register 16 dikeluarkan ke port C
```

Hasil simulasi:

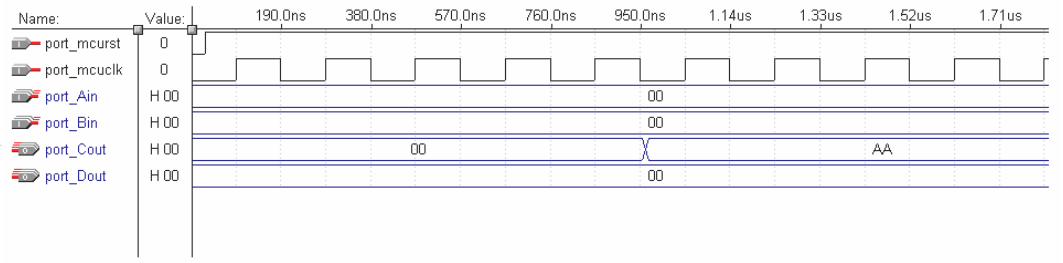


Mikrokontroler diinisialisasi dengan memberikan logika '0' pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi BREQ adalah sebagai berikut:

1. Register 16 diisi dengan data $AA.
2. Register 18 diisi dengan data $FF.
3. *Status register* diisi dengan isi dari register 18, yaitu $FF atau B 1111 1111.
4. Instruksi BREQ akan mengecek bilamana *flag* Z bernilai 1 maka percabangan dilakukan.
5. Instruksi selanjutnya yang dieksekusi adalah *out $15,r16* yang akan mengeluarkan isi register 16, yaitu $AA ke port C.

```
; Program uji instruksi BRMI (BRBS 2,k)
ldi r16,$AA          ; isi register 16 dengan data $AA
ldi r18,$FF          ; isi register 18 dengan data $FF
out $3F,r18          ; isi register 18 disimpan ke status register
brmi brmicabang      ; bila flag N=1 maka percabangan dilakukan ke brmicabang
out $15,r18          ; isi register 18 dikeluarkan ke port C
brmicabang: out $15,r16      ; isi register 16 dikeluarkan ke port C
```

Hasil simulasi:

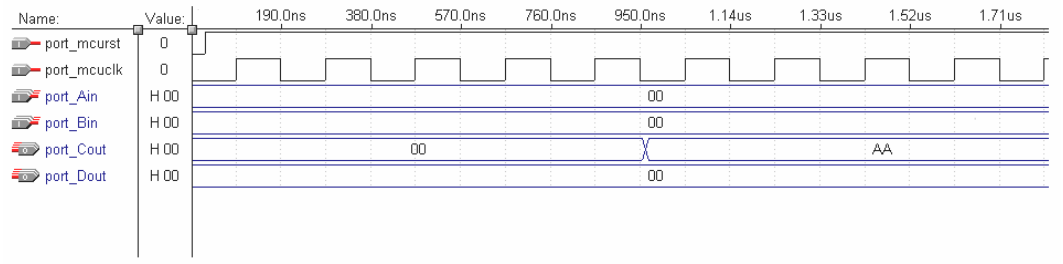


Mikrokontroler diinisialisasi dengan memberikan logika '0' pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi BRMI adalah sebagai berikut:

1. Register 16 diisi dengan data $AA.
2. Register 18 diisi dengan data $FF.
3. *Status register* diisi dengan isi dari register 18, yaitu $FF atau B 1111 1111.
4. Instruksi BRMI akan mengecek bilamana *flag* Z bernilai 1 maka percabangan dilakukan.
5. Instruksi selanjutnya yang dieksekusi adalah *out $15,r16* yang akan mengeluarkan isi register 16, yaitu $AA ke port C.

```
; Program uji instruksi BRVS (BRBS 3,k)
ldi r16,$AA          ; isi register 16 dengan data $AA
ldi r18,$FF          ; isi register 18 dengan data $FF
out $3F,r18          ; isi register 18 disimpan ke status register
brvs brvscabang      ; bila flag V=1 maka percabangan dilakukan ke brvscabang
out $15,r18          ; isi register 18 dikeluarkan ke port C
brvscabang: out $15,r16      ; isi register 16 dikeluarkan ke port C
```

Hasil simulasi:

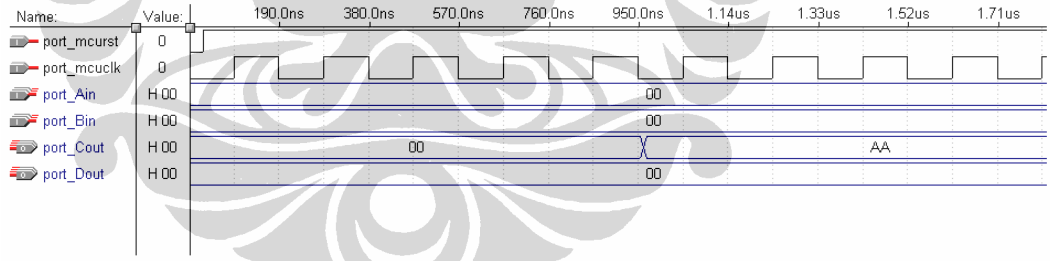


Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi BRVS adalah sebagai berikut:

1. Register 16 diisi dengan data $AA.
2. Register 18 diisi dengan data $FF.
3. *Status register* diisi dengan isi dari register 18, yaitu $FF atau B 1111 1111.
4. Instruksi BRVS akan mengecek bilamana *flag* V bernilai 1 maka percabangan dilakukan.
5. Instruksi selanjutnya yang dieksekusi adalah *out $15,r16* yang akan mengeluarkan isi register 16, yaitu $AA ke port C.

```
; Program uji instruksi BRLT (BRBS 4,k)
ldi r16,$AA             ; isi register 16 dengan data $AA
ldi r18,$FF             ; isi register 18 dengan data $FF
out $3F,r18             ; isi register 18 disimpan ke status register
brlt brltcabang         ; bila flag S=1 maka percabangan dilakukan ke brltcabang
out $15,r18             ; isi register 18 dikeluarkan ke port C
brltcabang: out $15,r16         ; isi register 16 dikeluarkan ke port C
```

Hasil simulasi:



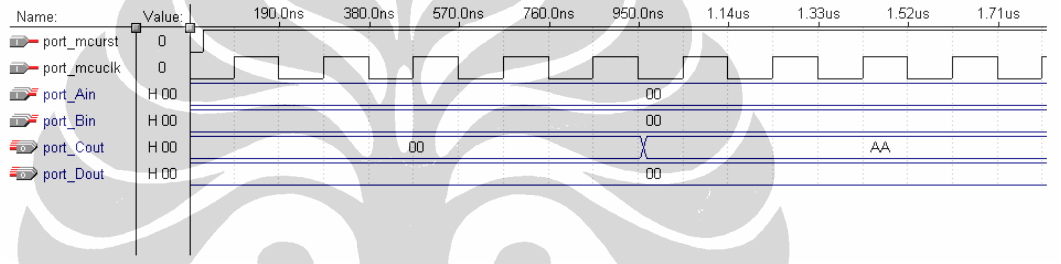
Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi BRLT adalah sebagai berikut:

1. Register 16 diisi dengan data $AA.
2. Register 18 diisi dengan data $FF.
3. *Status register* diisi dengan isi dari register 18, yaitu $FF atau B 1111 1111.
4. Instruksi BRLT akan mengecek bilamana *flag* S bernilai 1 maka percabangan dilakukan.

5. Instruksi selanjutnya yang dieksekusi adalah *out $15,r16* yang akan mengeluarkan isi register 16, yaitu $AA ke port C

```
; Program uji instruksi BRHS (BRBS 5,k)
ldi r16,$AA          ; isi register 16 dengan data $AA
ldi r18,$FF          ; isi register 18 dengan data $FF
out $3F,r18          ; isi register 18 disimpan ke status register
brhs brhscabang      ; bila flag H=1 maka percabangan dilakukan ke brhscabang
out $15,r18          ; isi register 18 dikeluarkan ke port C
brhscabang: out $15,r16      ; isi register 16 dikeluarkan ke port C
```

Hasil simulasi:

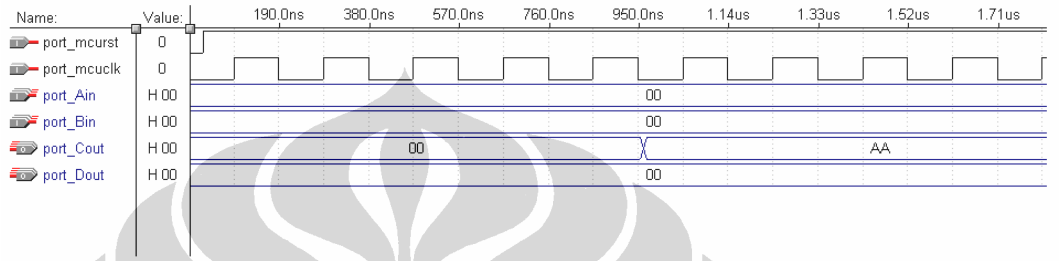


Mikrokontroler diinisialisasi dengan memberikan logika '0' pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi BRHS adalah sebagai berikut:

1. Register 16 diisi dengan data $AA.
2. Register 18 diisi dengan data $FF.
3. *Status register* diisi dengan isi dari register 18, yaitu $FF atau B 1111 1111.
4. Instruksi BRHS akan mengecek bilamana *flag* H bernilai 1 maka percabangan dilakukan.
5. Instruksi selanjutnya yang dieksekusi adalah *out $15,r16* yang akan mengeluarkan isi register 16, yaitu $AA ke port C.

```
; Program uji instruksi BRTS (BRBS 6,k)
ldi r16,$AA             ; isi register 16 dengan data $AA
ldi r18,$FF             ; isi register 18 dengan data $FF
out $3F,r18             ; isi register 18 disimpan ke status register
brvs brtscabang         ; bila flag T=1 maka percabangan dilakukan ke brtscabang
out $15,r18             ; isi register 18 dikeluarkan ke port C
brtscabang: out $15,r16         ; isi register 16 dikeluarkan ke port C
```

Hasil simulasi:

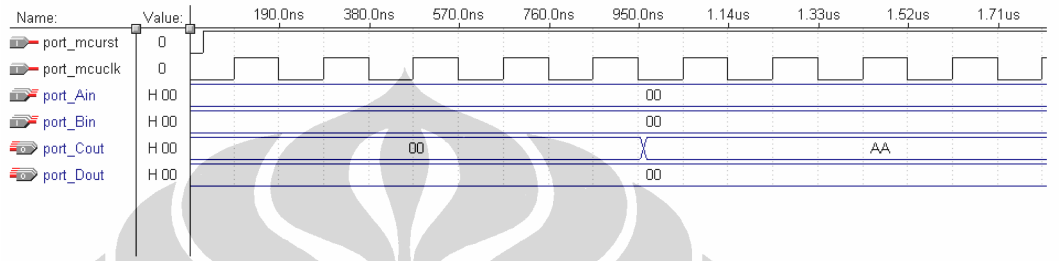


Mikrokontroler diinisialisasi dengan memberikan logika '0' pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi BRTS adalah sebagai berikut:

1. Register 16 diisi dengan data $AA.
2. Register 18 diisi dengan data $FF.
3. *Status register* diisi dengan isi dari register 18, yaitu $FF atau B 1111 1111.
4. Instruksi BRTS akan mengecek bilamana *flag* T bernilai 1 maka percabangan dilakukan.
5. Instruksi selanjutnya yang dieksekusi adalah *out $15,r16* yang akan mengeluarkan isi register 16, yaitu $AA ke port C.

```
; Program uji instruksi BRIE (BRBS 7,k)
ldi r16,$AA             ; isi register 16 dengan data $AA
ldi r18,$FF             ; isi register 18 dengan data $FF
out $3F,r18             ; isi register 18 disimpan ke status register
brie briecabang         ; bila flag I=1 maka percabangan dilakukan ke briecabang
out $15,r18             ; isi register 18 dikeluarkan ke port C
briecabang: out $15,r16         ; isi register 16 dikeluarkan ke port C
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika '0' pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi BRIE adalah sebagai berikut:

1. Register 16 diisi dengan data $AA.
2. Register 18 diisi dengan data $FF.
3. *Status register* diisi dengan isi dari register 18, yaitu $FF atau B 1111 1111.
4. Instruksi BRIE akan mengecek bilamana *flag* I bernilai 1 maka percabangan dilakukan.
5. Instruksi selanjutnya yang dieksekusi adalah *out $15,r16* yang akan mengeluarkan isi register 16, yaitu $AA ke port C.

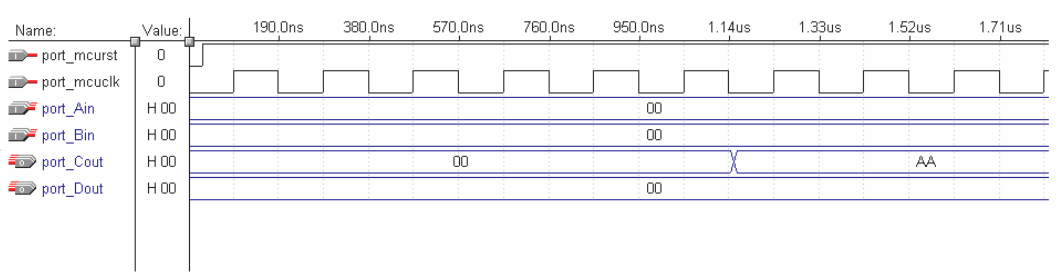
34. PENGUJIAN INSTRUKSI BRBC, BRCC, BRSH, BRNE, BRPL, BRVC, BRGE, BRHC, BRTC, BRID

Instruksi BRBC (*Branch if Status Flag Cleared*) mempunyai bentuk *BRBC s,k.* Instruksi ini akan melakukan percabangan bersyarat dengan mengevaluasi *flag* tertentu di *status register* (*if SREG(s)=0 then PC ß PC+k+1*). Percabangan akan dilakukan bila kondisi terpenuhi dan alamat percabangan adalah sejauh *k.* Pengujian terhadap instruksi ini sekaligus akan menguji instruksi *BRCC k* (*Branch if Carry Cleared*), *BRSH k* (*Branch if Same or Higher*), *BRNE k* (*Branch if Not Equal*), *BRPL k* (*Branch if Plus*), *BRVC k* (*Branch if Overflow Flag is Cleared*), *BRGE k* (*Branch if Greater or Equal, Signed*), *BRHC k* (*Branch if Half Carry Flag Cleared*), *BRTC k* (*Branch if T Flag Cleared*), dan *BRID k* (*Branch if Interrupt Disabled*). Instruksi *BRCC k* sama dengan instruksi *BRBC 0,k.* Instruksi *BRSH k* sama dengan instruksi *BRBC 0,k.* Instruksi *BRNE k* sama dengan instruksi *BRBC 1,k.* Instruksi *BRPL k* sama dengan instruksi *BRBC 2,k.* Instruksi *BRVC k* sama dengan instruksi *BRBC 3,k.* Instruksi *BRGE k* sama dengan instruksi *BRBC 4,k.* Instruksi *BRHC k* sama dengan instruksi *BRBC 5,k.* Instruksi *BRTC k* sama dengan instruksi *BRBC 6,k.* Instruksi *BRID k* sama dengan instruksi *BRBC 7,k.*

Pengujian kelompok instruksi ini dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Pengujian instruksi BRCC (BRBC 0,k)
ldi r16,$AA          ; register 16 diisi dengan data $AA
ldi r17,$00          ; register 17 diisi dengan data $00
ldi r18,$FF          ; register 18 diisi dengan data $FF
out $3F,r17          ; status register diisi dengan isi register 17
brcc brcccabang      ; percabangan dilakukan bila flag C=0
out $15,r18          ; isi register 18 dikeluarkan ke port C
brcccabang: out $15,r16      ; isi register 16 dikeluarkan ke port C
```

Hasil simulasi:

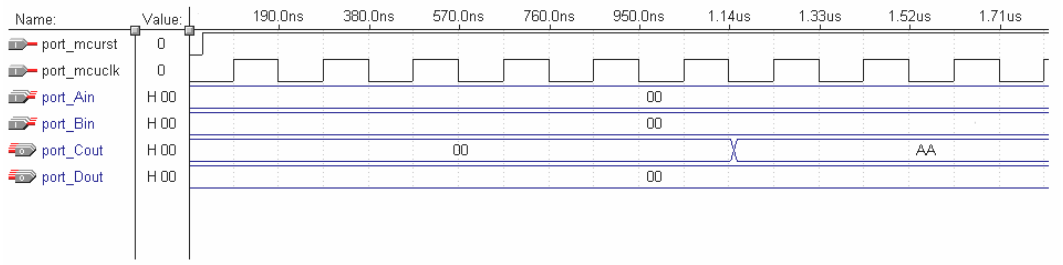


Mikrokontroler diinisialisasi dengan memberikan logika '0' pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi BRCC adalah sebagai berikut:

1. Register 16 diisi dengan data $AA.
2. Register 17 diisi dengan data $00.
3. Register 18 diisi dengan data $FF.
4. *Status register* diisi dengan isi dari register 17, yaitu $00.
5. Percabangan dilakukan ke instruksi *out $15,r16*, sehingga isi register 16, yaitu $AA, dikeluarkan ke port C.

```
; Pengujian instruksi BRSH (BRBC 0,k)
ldi r16,$AA             ; register 16 diisi dengan data $AA
ldi r17,$00             ; register 17 diisi dengan data $00
ldi r18,$FF             ; register 18 diisi dengan data $FF
out $3F,r17             ; status register diisi dengan isi register 17
brsh brshcabang         ; percabangan dilakukan bila flag C=0
out $15,r18             ; isi register 18 dikeluarkan ke port C
brshcabang: out $15,r16       ; isi register 16 dikeluarkan ke port C
```

Hasil simulasi:

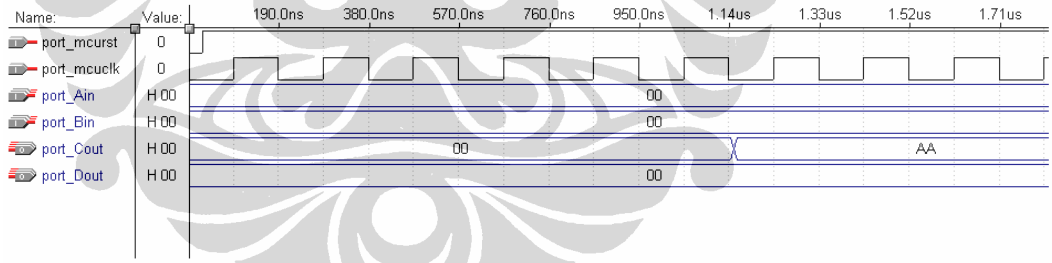


Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi BRSH adalah sebagai berikut:

1. Register 16 diisi dengan data $AA.
2. Register 17 diisi dengan data $00.
3. Register 18 diisi dengan data $FF.
4. *Status register* diisi dengan isi dari register 17, yaitu $00.
5. Percabangan dilakukan ke instruksi *out $15,r16*, sehingga isi register 16, yaitu $AA, dikeluarkan ke port C.

```
; Pengujian instruksi BRNE (BRBC 1,k)
ldi r16,$AA             ; register 16 diisi dengan data $AA
ldi r17,$00             ; register 17 diisi dengan data $00
ldi r18,$FF             ; register 18 diisi dengan data $FF
out $3F,r17             ; status register diisi dengan isi register 17
brne brnecabang         ; percabangan dilakukan bila flag Z=0
out $15,r18             ; isi register 18 dikeluarkan ke port C
brnecabang: out $15,r16       ; isi register 16 dikeluarkan ke port C
```

Hasil simulasi:



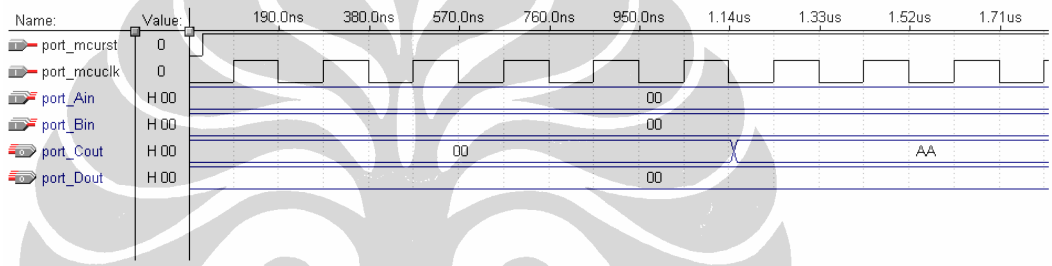
Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi BRNE adalah sebagai berikut:

1. Register 16 diisi dengan data $AA.
2. Register 17 diisi dengan data $00.
3. Register 18 diisi dengan data $FF.
4. *Status register* diisi dengan isi dari register 17, yaitu $00.

5. Percabangan dilakukan ke instruksi *out $15,r16*, sehingga isi register 16, yaitu $AA, dikeluarkan ke port C.

```
; Pengujian instruksi BRPL (BRBC 2,k)
ldi r16,$AA          ; register 16 diisi dengan data $AA
ldi r17,$00          ; register 17 diisi dengan data $00
ldi r18,$FF          ; register 18 diisi dengan data $FF
out $3F,r17          ; status register diisi dengan isi register 17
brpl brplcabang      ; percabangan dilakukan bila flag N=0
out $15,r18          ; isi register 18 dikeluarkan ke port C
brplcabang: out $15,r16      ; isi register 16 dikeluarkan ke port C
```

Hasil simulasi:

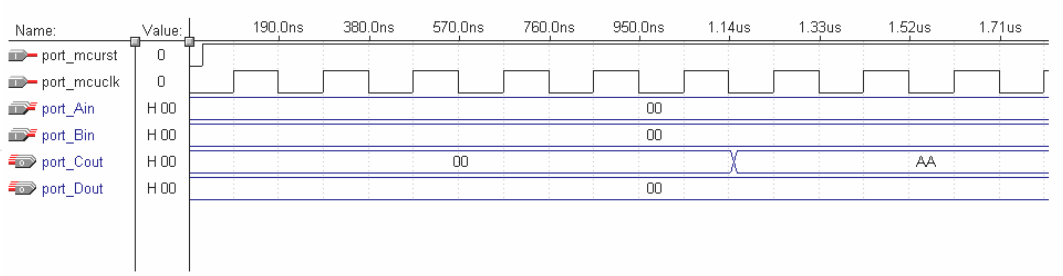


Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi BRPL adalah sebagai berikut:

1. Register 16 diisi dengan data $AA.
2. Register 17 diisi dengan data $00.
3. Register 18 diisi dengan data $FF.
4. *Status register* diisi dengan isi dari register 17, yaitu $00.
5. Percabangan dilakukan ke instruksi *out $15,r16*, sehingga isi register 16, yaitu $AA, dikeluarkan ke port C.

```
; Pengujian instruksi BRVC (BRBC 3,k)
ldi r16,$AA          ; register 16 diisi dengan data $AA
ldi r17,$00          ; register 17 diisi dengan data $00
ldi r18,$FF          ; register 18 diisi dengan data $FF
out $3F,r17          ; status register diisi dengan isi register 17
brvc brvccabang      ; percabangan dilakukan bila flag V=0
out $15,r18          ; isi register 18 dikeluarkan ke port C
brvccabang: out $15,r16      ; isi register 16 dikeluarkan ke port C
```

Hasil simulasi:

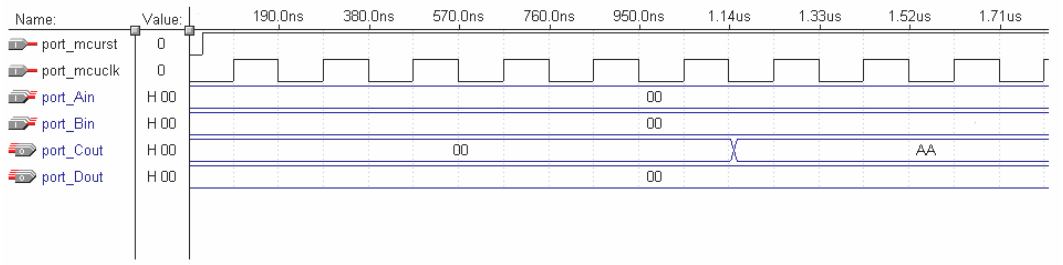


Mikrokontroler diinisialisasi dengan memberikan logika '0' pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi BRVC adalah sebagai berikut:

1. Register 16 diisi dengan data $AA.
2. Register 17 diisi dengan data $00.
3. Register 18 diisi dengan data $FF.
4. *Status register* diisi dengan isi dari register 17, yaitu $00.
5. Percabangan dilakukan ke instruksi *out $15,r16*, sehingga isi register 16, yaitu $AA, dikeluarkan ke port C.

```
; Pengujian instruksi BRGE (BRBC 4,k)
ldi r16,$AA             ; register 16 diisi dengan data $AA
ldi r17,$00             ; register 17 diisi dengan data $00
ldi r18,$FF             ; register 18 diisi dengan data $FF
out $3F,r17             ; status register diisi dengan isi register 17
brge brgecabang         ; percabangan dilakukan bila flag S=0
out $15,r18             ; isi register 18 dikeluarkan ke port C
brgecabang: out $15,r16       ; isi register 16 dikeluarkan ke port C
```

Hasil simulasi:

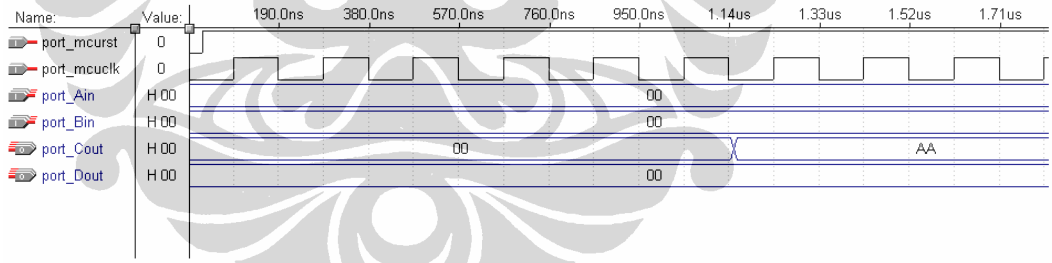


Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi BRGE adalah sebagai berikut:

1. Register 16 diisi dengan data $AA.
2. Register 17 diisi dengan data $00.
3. Register 18 diisi dengan data $FF.
4. *Status register* diisi dengan isi dari register 17, yaitu $00.
5. Percabangan dilakukan ke instruksi *out $15,r16*, sehingga isi register 16, yaitu $AA, dikeluarkan ke port C.

```
; Pengujian instruksi BRHC (BRBC 5,k)
ldi r16,$AA             ; register 16 diisi dengan data $AA
ldi r17,$00             ; register 17 diisi dengan data $00
ldi r18,$FF             ; register 18 diisi dengan data $FF
out $3F,r17             ; status register diisi dengan isi register 17
brhc brhccabang         ; percabangan dilakukan bila flag H=0
out $15,r18             ; isi register 18 dikeluarkan ke port C
brhccabang: out $15,r16       ; isi register 16 dikeluarkan ke port C
```

Hasil simulasi:



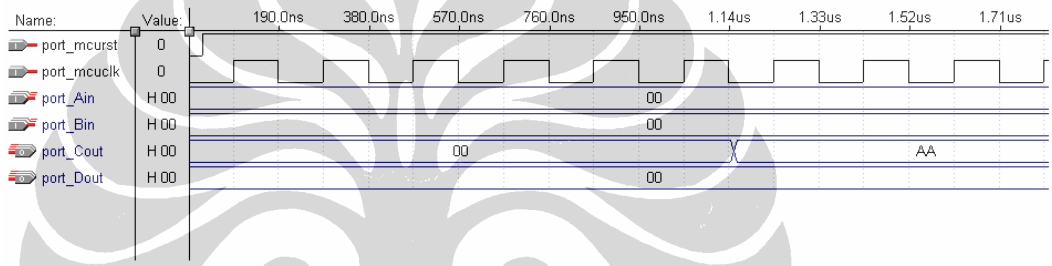
Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi BRHC adalah sebagai berikut:

1. Register 16 diisi dengan data $AA.
2. Register 17 diisi dengan data $00.
3. Register 18 diisi dengan data $FF.
4. *Status register* diisi dengan isi dari register 17, yaitu $00.

5. Percabangan dilakukan ke instruksi *out $15,r16*, sehingga isi register 16, yaitu $AA, dikeluarkan ke port C

```
; Pengujian instruksi BRTC (BRBC 6,k)
ldi r16,$AA          ; register 16 diisi dengan data $AA
ldi r17,$00          ; register 17 diisi dengan data $00
ldi r18,$FF          ; register 18 diisi dengan data $FF
out $3F,r17          ; status register diisi dengan isi register 17
brtc brtccabang      ; percabangan dilakukan bila flag T=0
out $15,r18          ; isi register 18 dikeluarkan ke port C
brtccabang: out $15,r16      ; isi register 16 dikeluarkan ke port C
```

Hasil simulasi:

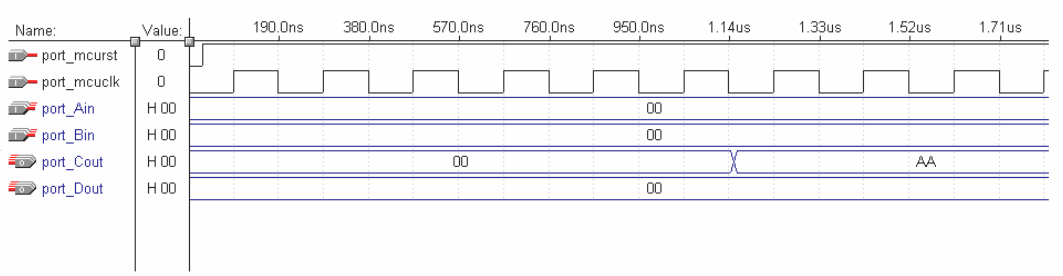


Mikrokontroler diinisialisasi dengan memberikan logika '0' pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi BRTC adalah sebagai berikut:

1. Register 16 diisi dengan data $AA.
2. Register 17 diisi dengan data $00.
3. Register 18 diisi dengan data $FF.
4. *Status register* diisi dengan isi dari register 17, yaitu $00.
5. Percabangan dilakukan ke instruksi *out $15,r16*, sehingga isi register 16, yaitu $AA, dikeluarkan ke port C.

```
; Pengujian instruksi BRID (BRBC 7,k)
ldi r16,$AA          ; register 16 diisi dengan data $AA
ldi r17,$00          ; register 17 diisi dengan data $00
ldi r18,$FF          ; register 18 diisi dengan data $FF
out $3F,r17          ; status register diisi dengan isi register 17
brid bridcabang      ; percabangan dilakukan bila flag I=0
out $15,r18          ; isi register 18 dikeluarkan ke port C
bridcabang: out $15,r16      ; isi register 16 dikeluarkan ke port C
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika '0' pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi BRID adalah sebagai berikut:

1. Register 16 diisi dengan data $AA.
2. Register 17 diisi dengan data $00.
3. Register 18 diisi dengan data $FF.
4. *Status register* diisi dengan isi dari register 17, yaitu $00.
5. Percabangan dilakukan ke instruksi *out $15,r16*, sehingga isi register 16, yaitu $AA, dikeluarkan ke port C.

35. PENGUJIAN INSTRUKSI CLR DAN SER

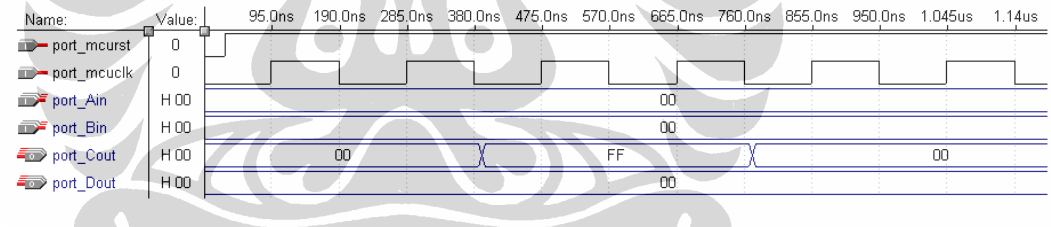
Instruksi CLR (*Clear Register*) mempunyai bentuk *CLR Rd.* Instruksi ini akan meng-*clear* semua bit pada sebuah register. Pada dasarnya instruksi ini melakukan operasi Exclusive OR antara register tersebut dengan dirinya sendiri (*Rd ß Rd* ⊕ *Rr*).

Instruksi SER (*Sets all Bits in Register*) mempunyai bentuk *SER Rd.* Instruksi ini akan men-*set* semua bit pada sebuah register atau mengisi register tersebut dengan nilai $FF (*Rd ß $FF*).

Pengujian instruksi CLR dan SER dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi SER dan CLR
ser r16         ; instruksi SER dilakukan terhadap isi register 16
out $15,r16     ; isi register 16 dikeluarkan pada port C dengan alamat $15
clr r16         ; instruksi CLR dilakukan terhadap isi register 16
out $15,r16     ; isi register 16 dikeluarkan pada port C dengan alamat $15
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi SER dan CLR adalah sebagai berikut:

1. Instruksi SER dilakukan terhadap register 16, sehingga semua bit di dalam register 16 menjadi ’1’.
2. Isi register 16 dikeluarkan ke port C dengan alamat $15.
3. Instruksi CLR dilakukan terhadap register 16 kembali, sehingga semua bit di dalam register 16 menjadi ’0’;
4. Isi register 16 dikeluarkan ke port C kembali.

36. PENGUJIAN INSTRUKSI CP DAN CPI

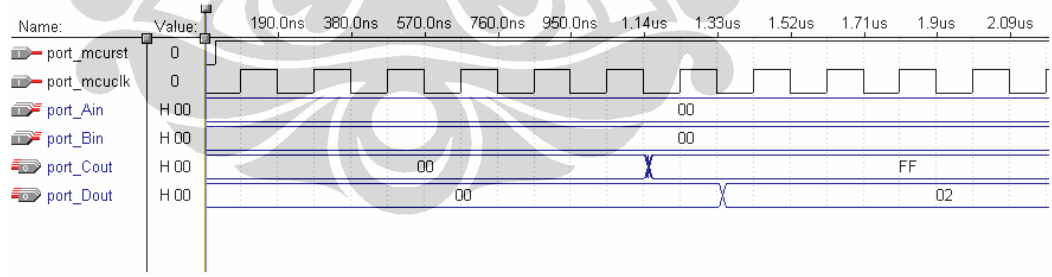
Instruksi CP (*Compare*) mempunyai bentuk *CP Rd,Rr*. Register Rd dan Rr tidak berubah setelah operasi dilakukan. Instruksi ini akan membandingkan antara isi register Rd dan isi register Rr apakah sama atau tidak. Semua instruksi percabangan bersyarat dapat digunakan setelah operasi instruksi ini.

Instruksi CPI (*Compare wih Immediate*) mempunyai bentuk *CPI Rd,K*. Instruksi ini akan membandingkan antara isi register Rd dengan sebuah konstanta K. Semua instruksi percabangan bersyarat juga dapat digunakan setelah operasi instruksi ini.

Pengujian instruksi CP dan CPI dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi CP
ldi r16,$02          ; register 16 diisi data $02
ldi r17,$02          ; register 17 diisi data $02
ldi r18,$FF          ; register 18 diisi data $FF
cp r16,r17           ; operasi CP antara isi register 16 dan isi register 17
brne lompat          ; percabangan bila tidak sama
out $15,r18          ; isi register 18 dikeluarkan ke port C
lompat: out $12,r16  ; isi register 16 dikeluarkan ke port D
```

Hasil simulasi:



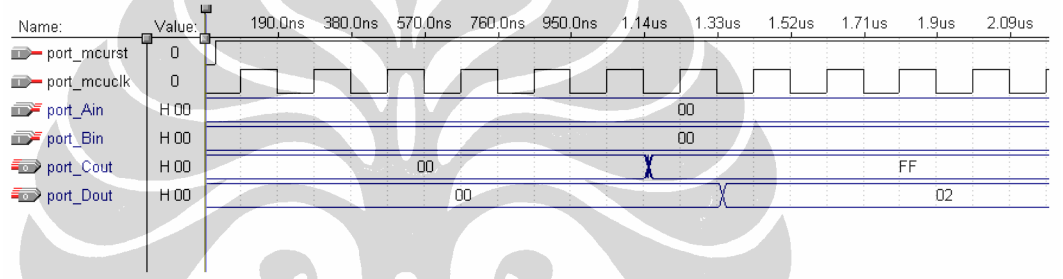
Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi CP adalah sebagai berikut:

1. Register 16 diisi data $02.
2. Register 17 diisi data $02.
3. Register 18 diisi data $FF.
4. Operasi CP dilakukan terhadap isi register 16 dan isi register 17.

5. Apabila isi register 16 dan isi register 17 sama, maka instruksi *out $15,r18* yang akan mengeluarkan isi register 18, yaitu $FF, ke port C.
6. Isi register 16, yaitu $02, dikeluarkan ke port D.

```
; Program uji instruksi CPI
ldi r16,$02          ; register 16 diisi data $02
ldi r18,$FF          ; register 18 diisi data $FF
cpi r16,$02          ; operasi CPI antara isi register 16 dan konstanta $02
brne lompat          ; percabangan bila tidak sama
out $15,r18          ; isi register 18 dikeluarkan ke port C
lompat: out $12,r16  ; isi register 16 dikeluarkan ke port D
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi CPI adalah sebagai berikut:

1. Register 16 diisi data $02.
2. Register 18 diisi data $FF.
3. Operasi CPI dilakukan terhadap isi register 16 dan konstanta $02.
4. Apabila isi register 16 dan konstanta sama, maka instruksi *out $15,r18* yang akan mengeluarkan isi register 18, yaitu $FF, ke port C.
5. Isi register 16, yaitu $02, dikeluarkan ke port D.

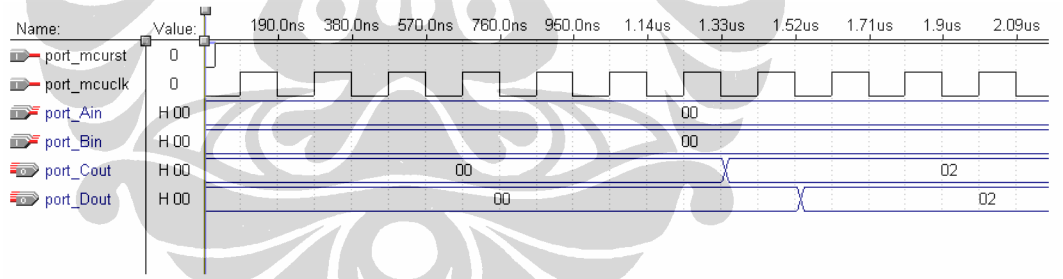
37. PENGUJIAN INSTRUKSI CPC

Instruksi CPC (*Compare with Carry*) mempunyai bentuk *CPC Rd,Rr*. Instruksi ini akan melakukan pembandingan antara isi register Rd dan isi register Rr serta *carry* yang tersimpan hasil operasi sebelumnya. Isi register Rd dan isi register Rr tidak berubah setelah operasi dilakukan. Semua instruksi percabangan bersyarat dapat digunakan setelah operasi instruksi CPC.

Pengujian instruksi CPC dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi CPC
ldi r16,$02          ; register 16 diisi data $02
ldi r17,$02          ; register 17 diisi data $02
ldi r18,$02          ; register 18 diisi data $02
cp r16,r17           ; operasi CP antara isi register 16 dan 17
cpc r16,r18          ; operasi CPC antara isi register 16 dan 18
brne lompat          ; operasi percabangan bila tidak sesuai
out $15,r18          ; isi register 18 dikeluarkan ke port C
lompat: out $12,r16  ; isi register 16 dikeluarkan ke port D
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi CPC adalah sebagai berikut:

1. Register 16 diisi data $02.
2. Register 17 diisi data $02.
3. Register 18 diisi data $02.
4. Operasi CP dilakukan antara isi register 16 dan isi register 17. Isi register 16 adalah sama dengan isi register 17, yaitu $02, sehingga hasil operasi ini akan menghasilkan *carry* ’0’.

5. Operasi CPC dilakukan antara isi register 16 dan isi register 18 serta *carry* hasil operasi sebelumnya. Hasil operasi ini akan menghasilkan *flag zero* 1 sehingga instruksi selanjutnya yang dieksekusi adalah instruksi *out $15,r18* yang akan mengeluarkan isi register 18, yaitu $02 ke port C.
6. Isi register 16 dikeluarkan ke port D.

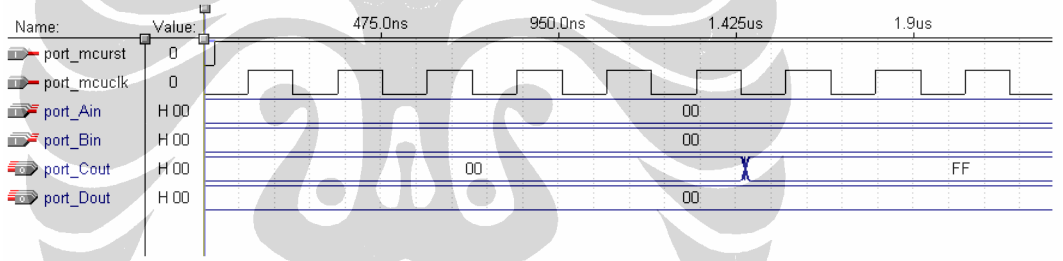
38. PENGUJIAN INSTRUKSI CPSE

Instruksi CPSE (*Compare, Skip if Equal*) mempunyai bentuk *CPSE Rd,Rr*. Instruksi ini melakukan pembandingan antara isi register Rd dan isi register Rr, dan akan melewati instruksi berikutnya bila isi register Rd dan isi register Rr sama.

Pengujian instruksi CPSE dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Pengujian instruksi CPSE
ldi r16,$AB     ; register 16 diisi data $AB
ldi r17,$AB     ; register 17 diisi data $AB
ldi r18,$FF     ; register 18 diisi data $FF
cpse r16,r17    ; operasi CPSE antara isi register 16 dan 17
out $12,r16     ; isi register 16 dikeluarkan ke port D
out $15,r18     ; isi register 18 dikeluarkan ke port C
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi CPSE adalah sebagai berikut:

1. Register 16 diisi data $AB.
2. Register 17 diisi data $AB.
3. Register 18 diisi data $FF.
4. Operasi CPSE antara isi register 16 dan isi register 17. Isi register 16 dan isi register 17 adalah sama, sehingga instruksi berikutnya, yaitu *out $12,r16* tidak dieksekusi.
5. Instruksi yang akan dieksekusi selanjutnya adalah instruksi *out $15,r18* yang akan mengeluarkan isi register 18, yaitu $FF, ke port C.

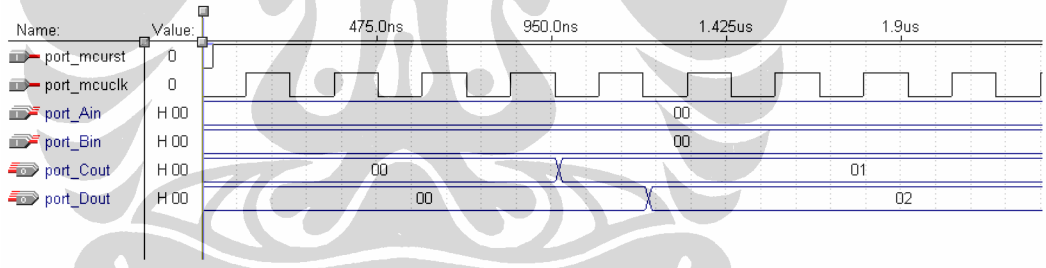
39. PENGUJIAN INSTRUKSI MOVW

Instruksi MOVW (*Copy Register Word*) mempunyai bentuk *MOVW Rd+1:Rd,Rr+1:Rr*. Instruksi ini akan mengkopi isi dari sepasang register ke sepasang register berikutnya (*Rd+1:Rd ß Rr+1:Rr*). Perancangan ini memakai register 2 dan register 3 sebagai pasangan register sumber dan register 0 dan register 1 sebagai pasangan register tujuan.

Pengujian instruksi MOVW dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Pengujian instruksi MOVW
ldi r18,$01          ; register 18 diisi data $01
ldi r19,$02          ; register 19 diisi data $02
movw r16,r18         ; operasi MOVW dengan register tujuan register 16 dan 17,
                     ; dan register sumber register 18 dan 19
out $15,r16          ; isi register 16 dikeluarkan ke port C
out $12,r17          ; isi register 17 dikeluarkan ke port D
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi MOVW adalah sebagai berikut:

1. Register 18 diisi data $01.
2. Register 19 diisi data $02.
3. Operasi MOVW akan memindahkan isi register 18 ke register 16, isi register 19 ke register 17.
4. Isi register 16 dikeluarkan ke port C.
5. Isi register 17 dikeluarkan ke port D.

40. PENGUJIAN INSTRUKSI PUSH DAN POP

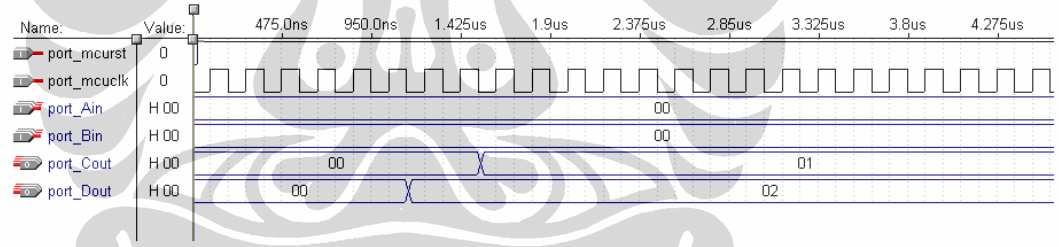
Instruksi PUSH (*Push Register on Stack*) mempunyai bentuk *PUSH Rr*. Instruksi ini akan menyimpan isi register Rr ke dalam *stack* (*stack ß Rr*).

Instruksi POP (*Pop Register from Stack*) mempunyai bentuk *POP Rd.* Instruksi ini akan mengambil isi *stack* ke dalam register Rr.

Pengujian instruksi PUSH dan POP dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Pengujian instruksi PUSH dan POP
ldi r16,$01     ; isi register dengan data $01
push r16        ; simpan isi register 16 ke stack
ldi r17,$02     ; isi register dengan data $02
push r17        ; simpan isi register 17 ke stack
pop r21         ; ambil isi stack ke register 21
out $12,r21     ; isi register 21 dikeluarkan ke port D
pop r22         ; ambil isi stack ke register 22
out $15,r22     ; isi register 22 dikeluarkan ke port C
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi PUSH dan POP adalah sebagai berikut:

1. Register 16 diisi dengan data $01.
2. Isi register 16 disimpan ke dalam *stack*.
3. Register 17 diisi dengan data $02.
4. Isi register 17 disimpan ke dalam *stack*.
5. Ambil isi *stack* dan simpan ke register 21.
6. Isi register 21 dikeluarkan ke port D.
7. Ambil isi *stack* dan simpan ke register 22.
8. Isi register 22 dikeluarkan ke port C.

41. PENGUJIAN INSTRUKSI SBIS DAN SBIC

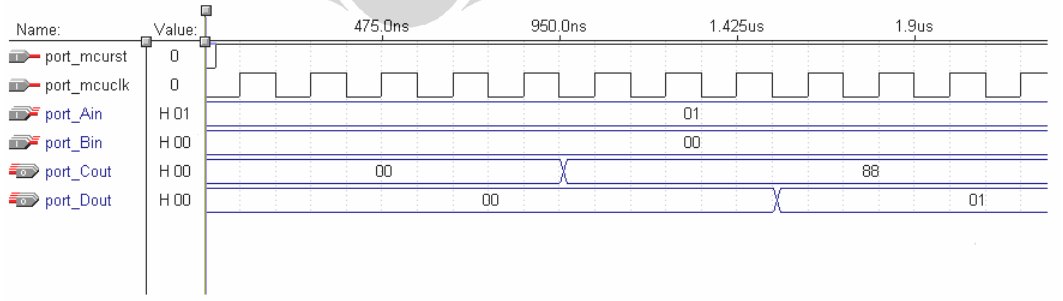
Instruksi SBIS (*Skip if Bit in I/O Register is Set*) mempunyai bentuk *SBIS A,b*. Instruksi ini melakukan tes pada sebuah bit dengan posisi *b* dari sebuah I/O register dengan alamat *A* dan akan melewati eksekusi instruksi berikutnya bilamana bit tersebut bernilai '1'.

Instruksi SBIC (*Skip if Bit in I/O Register is Cleared*) mempunyai bentuk *SBIC A,b*. Instruksi ini melakukan tes pada sebuah bit dengan posisi *b* dari sebuah I/O register dengan alamat *A* dan akan melewati eksekusi instruksi berikutnya bilamana bit tersebut bernilai '0'.

Perancangan ini menggunakan I/O register port A yang beralamat $19 dan port B yang beralamat $19. Pengujian instruksi SBIS dan SBIC dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi SBIS dan SBIC
; port A diberi data $01
ldi r18,$01     ; register 18 diisi data $01
ldi r19,$88     ; register 19 diisi data $88
sbis $19,0      ; pin A bit-0 set? skip bila bernilai '1'
out $15,r18     ; isi register 18 dikeluarkan ke port C
out $15,r19     ; isi register 19 dikeluarkan ke port C
sbic $19,1      ; pin A bit-1 clear? skip bila bernilai '0'
out $12,r19     ; isi register 19 dikeluarkan ke port D
out $12,r18     ; isi register 18 dikeluarkan ke port D
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi SBIS dan SBIC adalah sebagai berikut:

1. Port A diberi input $01 atau B 0000 0001.
2. Register 18 diisi data $01.
3. Register 19 diisi data $88.
4. Operasi SBIS dengan alamat I/O register $19 (port A) dan posisi bit 0. Instruksi selanjutnya yang dieksekusi adalah *out $15,r19* yang akan mengeluarkan isi register 19, yaitu $88, ke port C. Hal ini karena posisi bit 0 bernilai ’1’.
5. Operasi SBIC dengan alamat I/O register $19 (port A) dan posisi bit 1. Instruksi selanjutnya yang dieksekusi adalah *out $12,r18* yang akan mengeluarkan isi register 18, yaitu $01, ke port D. Hal ini karena posisi bit 1 bernilai ’0’.

42. PENGUJIAN INSTRUKSI SBRC DAN SBRS

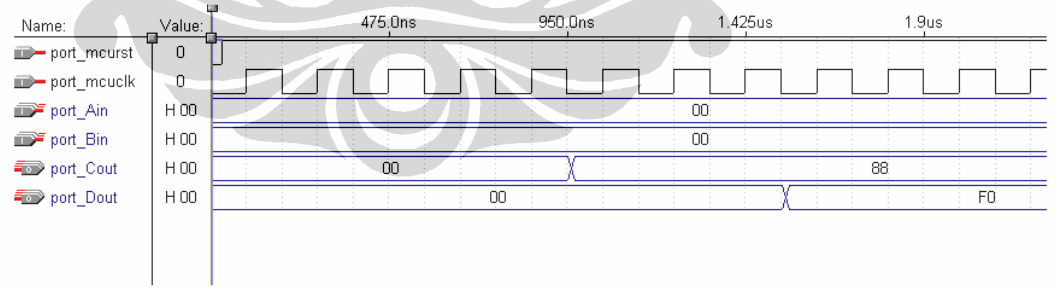
Instruksi SBRC (*Skip if Bit in Register Cleared*) mempunyai bentuk *SBRC Rr,b*. Instruksi ini melakukan tes terhadap isi register Rr dengan posisi bit b apakah bernilai ‘0’ atau tidak. Bila bernilai ’0’ maka instruksi berikutnya akan dilewati.

Instruksi SBRS (*Skip if Bit in Register is Set*) mempunyai bentuk *SBRS Rr,b*. Instruksi ini melakukan tes terhadap isi register Rr dengan posisi bit b apakah bernilai ‘1’ atau tidak. Bila bernilai ’1’ maka instruksi berikutnya akan dilewati.

Pengujian instruksi SBRC dan SBRS dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi SBRC dan SBRS
ldi r16,$F0     ; register 16 diisi data $F0
ldi r17,$88     ; register 17 diisi data $88
sbrc r16,0      ; bit ke-0 cleared? bila ya maka lewati instruksi berikutnya
out $15,r16     ; isi register 16 dikeluarkan ke port C
out $15,r17     ; isi register 17 dikeluarkan ke port C
sbrs r16,7      ; bit ke-7 set? bila ya maka lewati instruksi berikutnya
out $12,r17     ; isi register 17 dikeluarkan ke port D
out $12,r16     ; isi register 16 dikeluarkan ke port D
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi CPSE adalah sebagai berikut:

1. Register 16 diisi dengan data $F0.
2. Register 17 diisi dengan data $88.

3. Operasi SBRC terhadap isi register 16 posisi bit 0. Posisi bit 0 pada register 16 bernilai ’0’ sehingga instruksi selanjutnya tidak akan dieksekusi. Instruksi yang akan dieksekusi adalah *out $15,r17* yang akan mengeluarkan isi register 17, yaitu $88, ke port C.
4. Operasi SBRS terhadap isi register 16 posisi bit 7. Posisi bit 7 pada register 16 bernilai ’1’ sehingga instruksi selanjutnya tidak akan dieksekusi. Instruksi yang akan dieksekusi adalah *out $12,r16* yang akan mengeluarkan isi register 16, yaitu $F0, ke port D.

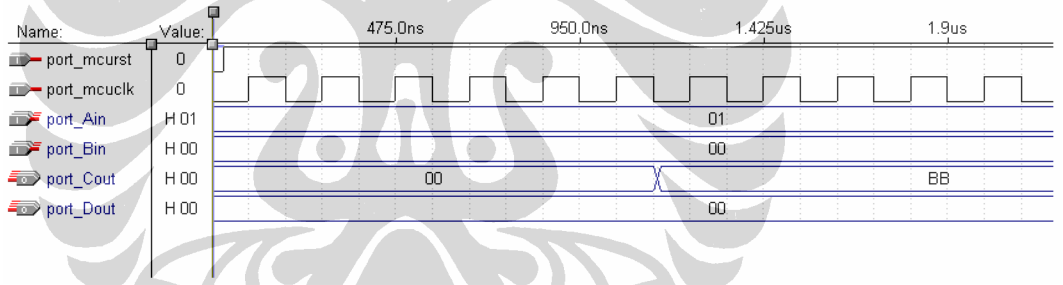
43. PENGUJIAN INSTRUKSI TST

Instruksi TST (*Test for Zero or Minus*) mempunyai bentuk *TST Rd*. Instruksi ini melakukan operasi logika AND terhadap isi register Rd. Isi register Rd tidak berubah setelah operasi.

Pengujian instruksi TST dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi TST
ldi r16,$00          ; register 16 diisi data $00
tst r16              ; operasi TST terhadap isi register 16
ldi r17,$AA          ; register 17 diisi data $AA
ldi r18,$BB          ; register 18 diisi data $BB
breq zero            ; percabangan ke zero bila flag zero='1'
out $15,r17          ; isi register 17 dikeluarkan ke port C
zero: out $15,r18    ; isi register 18 dikeluarkan ke port C
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika '0' pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi TST adalah sebagai berikut:

1. Register 16 diisi data $00.
2. Operasi TST dilakukan terhadap isi register 16.
3. Register 17 diisi data $AA.
4. Register 18 diisi data $BB.
5. Instruksi BREQ akan mengetes apakah *flag zero* bernilai '1'. Hasil dari operasi TST terhadap isi register 16 menghasilkan *flag zero* bernilai '1', sehingga percabangan dilakukan.
6. Instruksi out $15,r18 dieksekusi sehingga isi register 18, yaitu $BB, dikeluarkan ke port C.

44. PENGUJIAN INSTRUKSI ST DAN LD

Instruksi ST (*Store Indirect From Register to Data Space*) akan menyimpan sebuah *byte* dari sebuah register secara tidak langsung ke *data space*. Perancangan ini akan menggunakan RAM sebagai *data space*. Instruksi ST memakai isi dari register X, Y, atau Z sebagai alamat dari *data space*. Instruksi ini memiliki beberapa bentuk, yaitu:

ST X,Rr	(*(X) ß Rr*)
ST X+,Rr	(*(X) ß Rr;X ß X+1*)
ST -X,Rr	(*X ß X-1;(X) ß Rr*)
ST Y,Rr	(*(Y) ß Rr*)
ST Y+,Rr	(*(Y) ß Rr;Y ß Y+1*)
ST -Y,Rr	(*Y ß X-1;(Y) ß Rr*)
STD Y+q,Rr	(*(Y+q) ß Rr*)
ST Z,Rr	(*(Z) ß Rr*)
ST Z+,Rr	(*(Z) ß Rr;Z ß Z+1*)
ST -Z,Rr	(*Z ß Z-1;(Z) ß Rr*)
STD Z+q,Rr	(*(Z+q) ß Rr*)

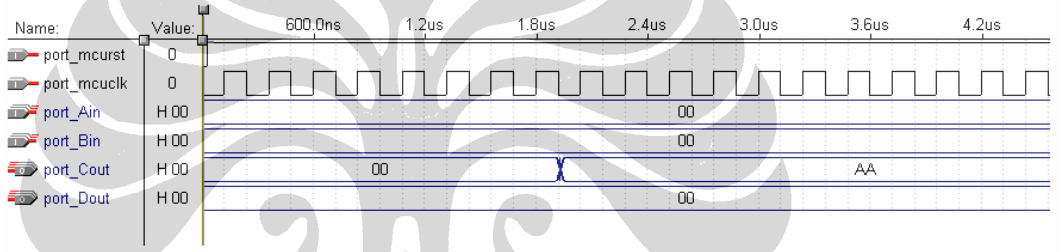
Instruksi LD (*Load Indirect from data space to Register*) mengambil isi dari *data space* dengan alamat yang ditunjuk oleh register X, Y, atau Z, dan hasilnya disimpan ke register Rd. Instruksi ini memiliki beberapa bentuk, yaitu:

LD Rd,X	(*Rd ß (X)*)
LD Rd,X+	(*Rd ß (X);X ß X+1*)
LD Rd,-X	(*X ß X-1;Rd ß (X)*)
LD Rd,Y	(*Rd ß (Y)*)
LD Rd,Y+	(*Rd ß (Y);Y ß Y+1*)
LD Rd,-Y	(*Y ß X-1;Rd ß (Y)*)
LDD Rd,Y+q	(*Rd ß (Y+q)*)
LD Rd,Z	(*Rd ß (Z)*)
LD Rd,Z+	(*Rd ß (Z);Z ß Z+1*)
LD Rd,-Z	(*Z ß Z-1;Rd ß (Z)*)
LDD Rd,Z+q	(*Rd ß (Z+q)*)

Pengujian instruksi ST dan LD akan dilakukan bersamaan, dimana instruksi ST akan menyimpan *byte* ke RAM dan instruksi LD akan mengambil *byte* dari RAM. Pengujian dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi ST dan LD dengan register X
; ST X,Rr dan LD Rd,X
ldi r26,$02      ; register X diisi data $02
ldi r16,$AA      ; register 16 diisi data $AA
st X,r16         ; simpan isi register 16 ke alamat yang ditunjuk register X
ld r17,X         ; ambil dari alamat yang ditunjuk register X ke register 17
out $15,r17      ; isi register 17 dikeluarkan ke port C
```

Hasil simulasi:

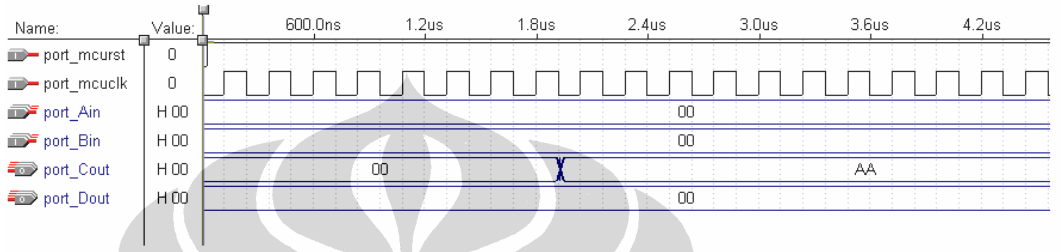


Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian tersebut adalah sebagai berikut:

1. Register 26 atau pointer X diisi data $02.
2. Register 16 diisi data $AA.
3. Instruksi ST akan menyimpan isi register 16 ke alamat RAM yang merupakan isi dari pointer X, yaitu $02.
4. Instruksi LD akan mengambil isi RAM, yaitu $AA, dengan alamat dari isi pointer X, yaitu $02 ke register 17.
5. Isi register 17 dikeluarkan ke port C.

```
; Program uji instruksi ST dan LD dengan register Y
; ST Y,Rr dan LD Rd,Y
ldi r28,$02     ; register Y diisi data $02
ldi r16,$AA     ; register 16 diisi data $AA
st Y,r16        ; simpan isi register 16 ke alamat yang ditunjuk register Y
ld r17,Y        ; ambil dari alamat yang ditunjuk register Y ke register 17
out $15,r17     ; isi register 17 dikeluarkan ke port C
```

Hasil simulasi:

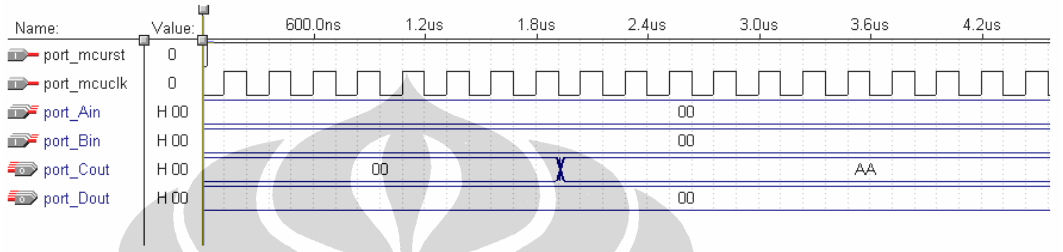


Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian tersebut adalah sebagai berikut:

1. Register 28 atau pointer Y diisi data $02.
2. Register 16 diisi data $AA.
3. Instruksi ST akan menyimpan isi register 16 ke alamat RAM yang merupakan isi dari pointer Y, yaitu $02.
4. Instruksi LD akan mengambil isi RAM, yaitu $AA, dengan alamat dari isi pointer Y, yaitu $02 ke register 17.
5. Isi register 17 dikeluarkan ke port C.

```
; Program uji instruksi ST dan LD dengan register Z
; ST Z,Rr dan LD Rd,Z
ldi r30,$02      ; register Z diisi data $02
ldi r16,$AA      ; register 16 diisi data $AA
st Z,r16         ; simpan isi register 16 ke alamat yang ditunjuk register Z
ld r17,Z         ; ambil dari alamat yang ditunjuk register Z ke register 17
out $15,r17      ; isi register 17 dikeluarkan ke port C
```

Hasil simulasi:

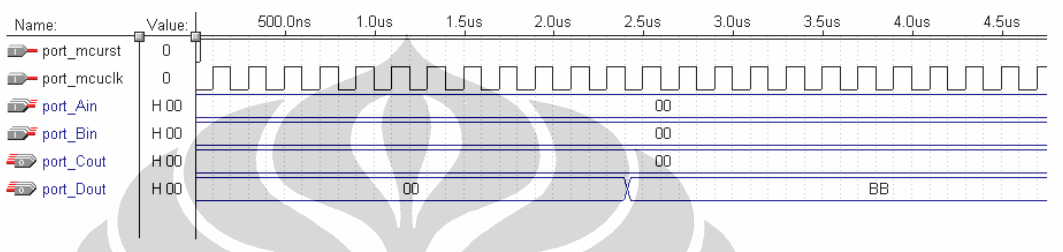


Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian tersebut adalah sebagai berikut:

1. Register 30 atau pointer Z diisi data $02.
2. Register 16 diisi data $AA.
3. Instruksi ST akan menyimpan isi register 16 ke alamat RAM yang merupakan isi dari pointer Z, yaitu $02.
4. Instruksi LD akan mengambil isi RAM, yaitu $AA, dengan alamat dari isi pointer Z, yaitu $02 ke register 17.
5. Isi register 17 dikeluarkan ke port C.

```
; Program uji instruksi ST dan LD dengan register X
; ST X+,Rr dan LD Rd,X
ldi r26,$02      ; register X diisi data $02
ldi r16,$BB      ; register 16 diisi data $BB
st X+,r16        ; simpan isi register 16 ke alamat yang ditunjuk register X ($02)
st X,r16         ; simpan isi register 16 ke alamat yang ditunjuk register X ($03)
ld r17,X         ; ambil dari alamat yang ditunjuk register X ($03) ke register 17
out $12,r17      ; isi register 17 dikeluarkan ke port D
```

Hasil simulasi:

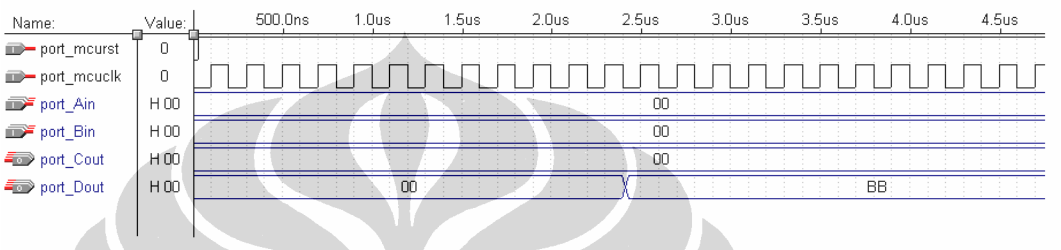


Mikrokontroler diinisialisasi dengan memberikan logika '0' pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian tersebut adalah sebagai berikut:

1. Register 26 atau pointer X diisi data $02.
2. Register 16 diisi data $BB.
3. Instruksi ST akan menyimpan isi register 16 ke alamat RAM yang merupakan isi dari pointer X, yaitu $02. Isi register 26 ditambah 1 sehingga menjadi $03.
4. Instruksi ST akan menyimpan isi register 16 ke alamat RAM yang merupakan isi dari pointer X, yaitu $03.
5. Instruksi LD akan mengambil isi RAM, yaitu $BB, dengan alamat dari isi pointer X, yaitu $03 ke register 17.
6. Isi register 17 dikeluarkan ke port D.

```
; Program uji instruksi ST dan LD dengan register Y
; ST Y+,Rr dan LD Rd,Y
ldi r28,$02     ; register Y diisi data $02
ldi r16,$BB     ; register 16 diisi data $BB
st Y+,r16       ; simpan isi register 16 ke alamat yang ditunjuk register Y ($02)
st Y,r16        ; simpan isi register 16 ke alamat yang ditunjuk register Y ($03)
ld r17,Y        ; ambil dari alamat yang ditunjuk register Y ($03) ke register 17
out $12,r17     ; isi register 17 dikeluarkan ke port D
```

Hasil simulasi:

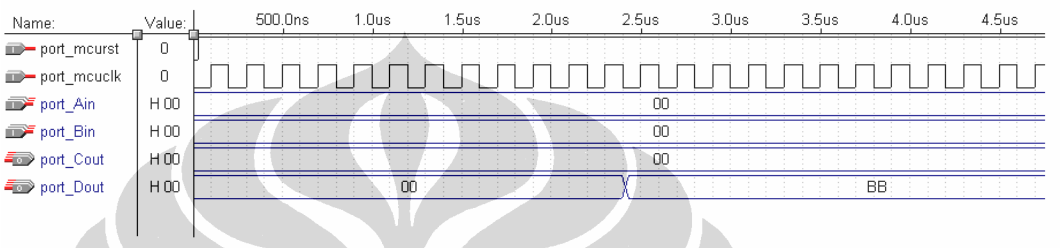


Mikrokontroler diinisialisasi dengan memberikan logika '0' pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian tersebut adalah sebagai berikut:

1. Register 28 atau pointer Y diisi data $02.
2. Register 16 diisi data $BB.
3. Instruksi ST akan menyimpan isi register 16 ke alamat RAM yang merupakan isi dari pointer Y, yaitu $02. Isi register 28 ditambah 1 sehingga menjadi $03.
4. Instruksi ST akan menyimpan isi register 16 ke alamat RAM yang merupakan isi dari pointer Y, yaitu $03.
5. Instruksi LD akan mengambil isi RAM, yaitu $BB, dengan alamat dari isi pointer Y, yaitu $03 ke register 17.
6. Isi register 17 dikeluarkan ke port D.

```
; Program uji instruksi ST dan LD dengan register Z
; ST Z+,Rr dan LD Rd,Z
ldi r30,$02     ; register Z diisi data $02
ldi r16,$BB     ; register 16 diisi data $BB
st Z+,r16       ; simpan isi register 16 ke alamat yang ditunjuk register Z ($02)
st Z,r16        ; simpan isi register 16 ke alamat yang ditunjuk register Z ($03)
ld r17,Z        ; ambil dari alamat yang ditunjuk register Z ($03) ke register 17
out $12,r17     ; isi register 17 dikeluarkan ke port D
```

Hasil simulasi:

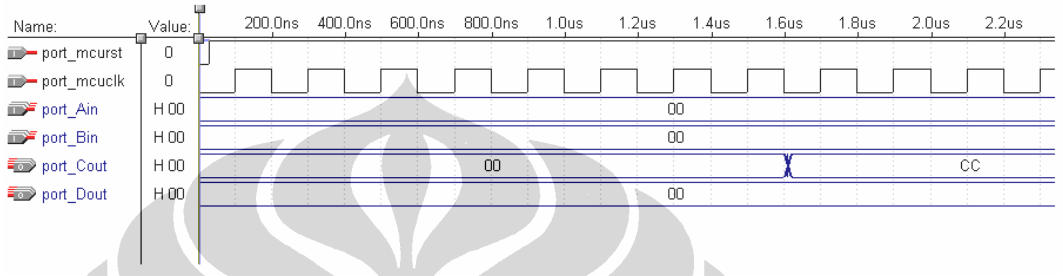


Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian tersebut adalah sebagai berikut:

1. Register 30 atau pointer Z diisi data $02.
2. Register 16 diisi data $BB.
3. Instruksi ST akan menyimpan isi register 16 ke alamat RAM yang merupakan isi dari pointer Z, yaitu $02. Isi register 30 ditambah 1 sehingga menjadi $03.
4. Instruksi ST akan menyimpan isi register 16 ke alamat RAM yang merupakan isi dari pointer Z, yaitu $03.
5. Instruksi LD akan mengambil isi RAM, yaitu $BB, dengan alamat dari isi pointer Z, yaitu $03 ke register 17.
6. Isi register 17 dikeluarkan ke port D.

```
; Program uji instruksi ST dan LD dengan register X
; ST -X,Rr dan LD Rd,X
ldi r26,$02      ; register X diisi data $02
ldi r16,$CC      ; register 16 diisi data $CC
st -X,r16        ; simpan isi register 16 ke alamat yang ditunjuk register X ($01)
ld r17,X         ; ambil dari alamat yang ditunjuk register X ($01) ke register 17
out $15,r17      ; isi register 17 dikeluarkan ke port C
```

Hasil simulasi:

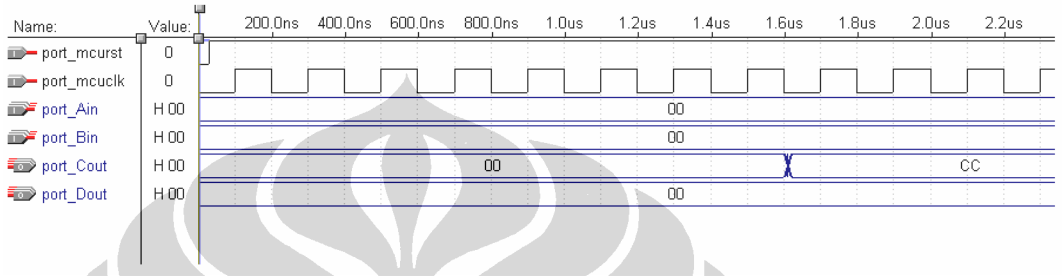


Mikrokontroler diinisialisasi dengan memberikan logika '0' pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian tersebut adalah sebagai berikut:

1. Register 26 atau pointer X diisi data $02.
2. Register 16 diisi data $CC.
3. Isi register 26 dikurang 1 sehingga menjadi $01. Instruksi ST akan menyimpan isi register 16 ke alamat RAM yang merupakan isi dari pointer X, yaitu $01.
4. Instruksi LD akan mengambil isi RAM, yaitu $CC, dengan alamat dari isi pointer X, yaitu $01 ke register 17.
5. Isi register 17 dikeluarkan ke port C.

```
; Program uji instruksi ST dan LD dengan register Y
; ST -Y,Rr dan LD Rd,Y
ldi r28,$02      ; register Y diisi data $02
ldi r16,$CC      ; register 16 diisi data $CC
st -Y,r16        ; simpan isi register 16 ke alamat yang ditunjuk register Y ($01)
ld r17,Y         ; ambil dari alamat yang ditunjuk register Y ($01) ke register 17
out $15,r17      ; isi register 17 dikeluarkan ke port C
```

Hasil simulasi:

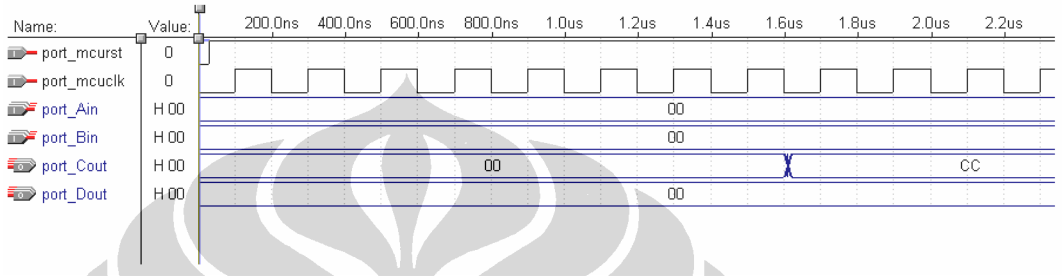


Mikrokontroler diinisialisasi dengan memberikan logika '0' pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian tersebut adalah sebagai berikut:

1. Register 28 atau pointer Y diisi data $02.
2. Register 16 diisi data $CC.
3. Isi register 28 dikurang 1 sehingga menjadi $01. Instruksi ST akan menyimpan isi register 16 ke alamat RAM yang merupakan isi dari pointer Y, yaitu $01.
4. Instruksi LD akan mengambil isi RAM, yaitu $CC, dengan alamat dari isi pointer Y, yaitu $01 ke register 17.
5. Isi register 17 dikeluarkan ke port C.

```
; Program uji instruksi ST dan LD dengan register Z
; ST -Z,Rr dan LD Rd,Z
ldi r30,$02      ; register Z diisi data $02
ldi r16,$CC      ; register 16 diisi data $CC
st -Z,r16        ; simpan isi register 16 ke alamat yang ditunjuk register Z ($01)
ld r17,Z         ; ambil dari alamat yang ditunjuk register Z ($01) ke register 17
out $15,r17      ; isi register 17 dikeluarkan ke port C
```

Hasil simulasi:

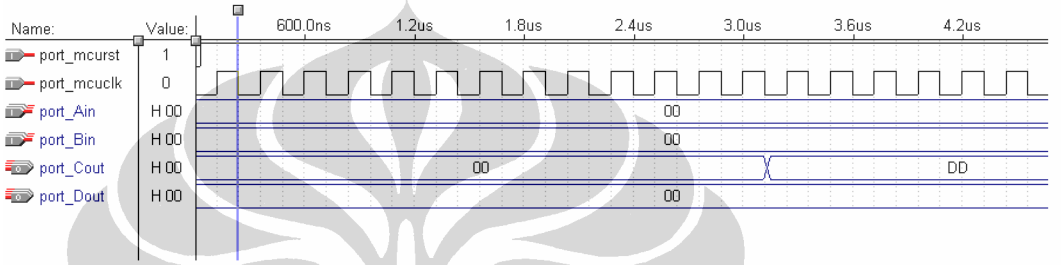


Mikrokontroler diinisialisasi dengan memberikan logika '0' pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian tersebut adalah sebagai berikut:

1. Register 30 atau pointer Z diisi data $02.
2. Register 16 diisi data $CC.
3. Isi register 30 dikurang 1 sehingga menjadi $01. Instruksi ST akan menyimpan isi register 16 ke alamat RAM yang merupakan isi dari pointer Z, yaitu $01.
4. Instruksi LD akan mengambil isi RAM, yaitu $CC, dengan alamat dari isi pointer Z, yaitu $01 ke register 17.
5. Isi register 17 dikeluarkan ke port C.

```
; Program uji instruksi ST dan LD dengan register X
; LD Rd,-X
ldi r26,$02      ; register X diisi data $02
ldi r16,$DD      ; register 16 diisi data $DD
ldi r17,$EE      ; register 17 diisi data $EE
st X+,r16        ; simpan isi register 16 ke alamat yang ditunjuk register X ($02)
st X,r17         ; simpan isi register 16 ke alamat yang ditunjuk register X ($03)
ld r17,-X        ; ambil dari alamat yang ditunjuk register X ($02) ke register 17
out $15,r17      ; isi register 17 dikeluarkan ke port C
```

Hasil simulasi:

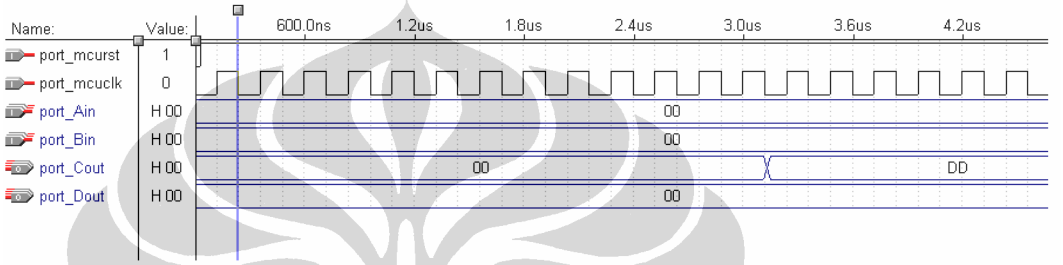


Mikrokontroler diinisialisasi dengan memberikan logika '0' pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian tersebut adalah sebagai berikut:

1. Register 26 atau pointer X diisi data $02.
2. Register 16 diisi data $DD.
3. Register 17 diisi data $EE.
4. Instruksi ST akan menyimpan isi register 16 ($DD) ke alamat RAM yang merupakan isi dari pointer X, yaitu $02. Isi register 26 ditambah 1 sehingga menjadi $03.
5. Instruksi ST akan menyimpan isi register 17 ($EE) ke alamat RAM yang merupakan isi dari pointer X, yaitu $03.
6. Isi register 26 dikurang 1 sehingga menjadi $02. Instruksi LD akan mengambil isi RAM, yaitu $DD, dengan alamat dari isi pointer X, yaitu $02 ke register 17.
7. Isi register 17 dikeluarkan ke port C.

```
; Program uji instruksi ST dan LD dengan register Y
; LD Rd,-Y
ldi r28,$02     ; register Y diisi data $02
ldi r16,$DD     ; register 16 diisi data $DD
ldi r17,$EE     ; register 17 diisi data $EE
st Y+,r16       ; simpan isi register 16 ke alamat yang ditunjuk register Y ($02)
st Y,r17        ; simpan isi register 16 ke alamat yang ditunjuk register Y ($03)
ld r17,-Y       ; ambil dari alamat yang ditunjuk register Y ($02) ke register 17
out $15,r17     ; isi register 17 dikeluarkan ke port C
```

Hasil simulasi:

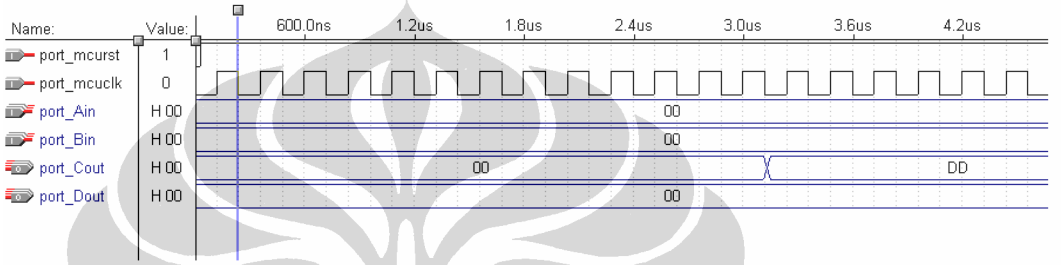


Mikrokontroler diinisialisasi dengan memberikan logika '0' pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian tersebut adalah sebagai berikut:

1. Register 28 atau pointer Y diisi data $02.
2. Register 16 diisi data $DD.
3. Register 17 diisi data $EE.
4. Instruksi ST akan menyimpan isi register 16 ($DD) ke alamat RAM yang merupakan isi dari pointer Y, yaitu $02. Isi register 28 ditambah 1 sehingga menjadi $03.
5. Instruksi ST akan menyimpan isi register 17 ($EE) ke alamat RAM yang merupakan isi dari pointer Y, yaitu $03.
6. Isi register 28 dikurang 1 sehingga menjadi $02. Instruksi LD akan mengambil isi RAM, yaitu $DD, dengan alamat dari isi pointer Y, yaitu $02 ke register 17.
7. Isi register 17 dikeluarkan ke port C.

```
; Program uji instruksi ST dan LD dengan register Z
; LD Rd,-Z
ldi r30,$02     ; register Z diisi data $02
ldi r16,$DD     ; register 16 diisi data $DD
ldi r17,$EE     ; register 17 diisi data $EE
st Z+,r16       ; simpan isi register 16 ke alamat yang ditunjuk register Z ($02)
st Z,r17        ; simpan isi register 16 ke alamat yang ditunjuk register Z ($03)
ld r17,-Z       ; ambil dari alamat yang ditunjuk register Z ($02) ke register 17
out $15,r17     ; isi register 17 dikeluarkan ke port C
```

Hasil simulasi:

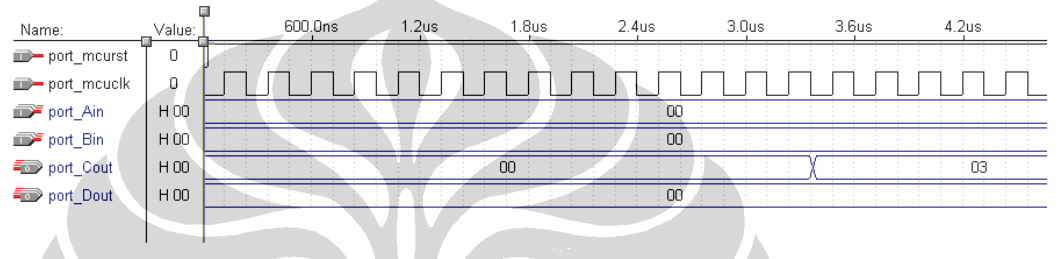


Mikrokontroler diinisialisasi dengan memberikan logika '0' pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian tersebut adalah sebagai berikut:

1. Register 30 atau pointer Z diisi data $02.
2. Register 16 diisi data $DD.
3. Register 17 diisi data $EE.
4. Instruksi ST akan menyimpan isi register 16 ($DD) ke alamat RAM yang merupakan isi dari pointer Z, yaitu $02. Isi register 30 ditambah 1 sehingga menjadi $03.
5. Instruksi ST akan menyimpan isi register 17 ($EE) ke alamat RAM yang merupakan isi dari pointer Z, yaitu $03.
6. Isi register 26 dikurang 1 sehingga menjadi $02. Instruksi LD akan mengambil isi RAM, yaitu $DD, dengan alamat dari isi pointer Z, yaitu $02 ke register 17.
7. Isi register 17 dikeluarkan ke port C.

```
; Program uji instruksi ST dan LD dengan register X
; LD Rd,X+
ldi r26,$02     ; register X diisi data $02
ldi r16,$DD     ; register 16 diisi data $DD
st X+,r16       ; simpan isi register 16 ke alamat yang ditunjuk register X ($02)
st X,r26        ; simpan isi register 26 ke alamat yang ditunjuk register X ($03)
ldi r26,$02     ; register 26 diisi $02
ld r17,X+       ; ambil dari alamat yang ditunjuk register X ($02) ke register 17
ld r18,X        ; ambil dari alamat yang ditunjuk register X ($03) ke register 18
out $15,r18     ; isi register 18 dikeluarkan ke port C
```

Hasil simulasi:

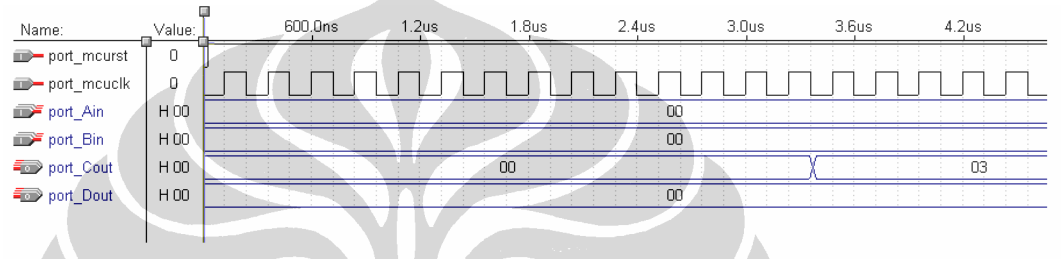


Mikrokontroler diinisialisasi dengan memberikan logika '0' pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian tersebut adalah sebagai berikut:

1. Register 26 atau pointer X diisi data $02.
2. Register 16 diisi data $DD.
3. Instruksi ST akan menyimpan isi register 16 ($DD) ke alamat RAM yang merupakan isi dari pointer X, yaitu $02. Isi register 26 ditambah 1 sehingga menjadi $03.
4. Instruksi ST akan menyimpan isi register 26 ($03) ke alamat RAM yang merupakan isi dari pointer X, yaitu $03.
5. Register 26 diisi $02.
6. Register 17 diisi data ($DD) yang tersimpan di alamat RAM yang tersimpan di register X. Isi register X ditambah 1 sehingga menjadi $03.
7. Register 18 diisi data ($03) yang tersimpan di alamat RAM yang tersimpan di register X.
8. Isi dari register 18 ($03) dikeluarkan ke port C.

```
; Program uji instruksi ST dan LD dengan register Y
; LD Rd,Y+
ldi r28,$02     ; register Y diisi data $02
ldi r16,$DD     ; register 16 diisi data $DD
st Y+,r16       ; simpan isi register 16 ke alamat yang ditunjuk register Y ($02)
st Y,r28        ; simpan isi register 28 ke alamat yang ditunjuk register Y ($03)
ldi r26,$02     ; register 26 diisi $02
ld r17,Y+       ; ambil dari alamat yang ditunjuk register Y ($02) ke register 17
ld r18,Y        ; ambil dari alamat yang ditunjuk register Y ($03) ke register 18
out $15,r18     ; isi register 18 dikeluarkan ke port C
```

Hasil simulasi:

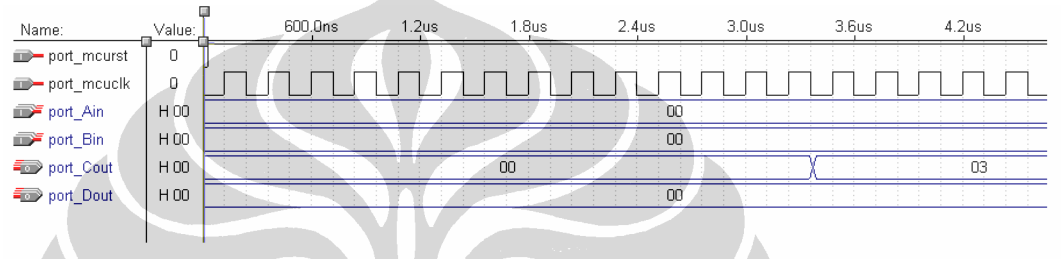


Mikrokontroler diinisialisasi dengan memberikan logika '0' pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian tersebut adalah sebagai berikut:

1. Register 28 atau pointer Y diisi data $02.
2. Register 16 diisi data $DD.
3. Instruksi ST akan menyimpan isi register 16 ($DD) ke alamat RAM yang merupakan isi dari pointer Y, yaitu $02. Isi register 28 ditambah 1 sehingga menjadi $03.
4. Instruksi ST akan menyimpan isi register 28 ($03) ke alamat RAM yang merupakan isi dari pointer Y, yaitu $03.
5. Register 28 diisi $02.
6. Register 17 diisi data ($DD) yang tersimpan di alamat RAM yang tersimpan di register Y. Isi register Y ditambah 1 sehingga menjadi $03.
7. Register 18 diisi data ($03) yang tersimpan di alamat RAM yang tersimpan di register Y.
8. Isi dari register 18 ($03) dikeluarkan ke port C.

```
; Program uji instruksi ST dan LD dengan register Z
; LD Rd,Z+
ldi r30,$02     ; register Z diisi data $02
ldi r16,$DD     ; register 16 diisi data $DD
st Z+,r16       ; simpan isi register 16 ke alamat yang ditunjuk register Z ($02)
st Z,r28        ; simpan isi register 28 ke alamat yang ditunjuk register Z ($03)
ldi r30,$02     ; register 30 diisi $02
ld r17,Z+       ; ambil dari alamat yang ditunjuk register Z ($02) ke register 17
ld r18,Z        ; ambil dari alamat yang ditunjuk register Z ($03) ke register 18
out $15,r18     ; isi register 18 dikeluarkan ke port C
```

Hasil simulasi:

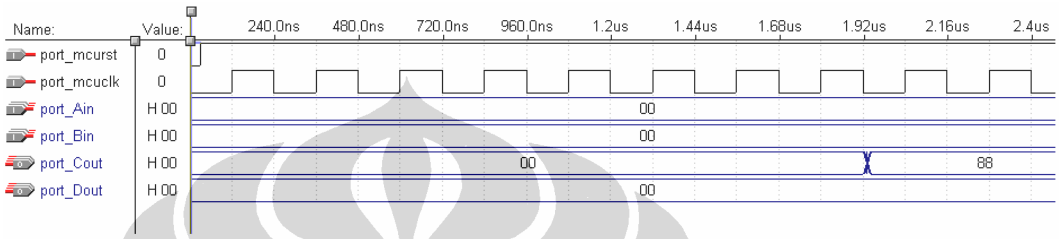


Mikrokontroler diinisialisasi dengan memberikan logika '0' pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian tersebut adalah sebagai berikut:

1. Register 30 atau pointer Z diisi data $02.
2. Register 16 diisi data $DD.
3. Instruksi ST akan menyimpan isi register 16 ($DD) ke alamat RAM yang merupakan isi dari pointer Z, yaitu $02. Isi register 30 ditambah 1 sehingga menjadi $03.
4. Instruksi ST akan menyimpan isi register 30 ($03) ke alamat RAM yang merupakan isi dari pointer Z, yaitu $03.
5. Register 30 diisi $02.
6. Register 17 diisi data ($DD) yang tersimpan di alamat RAM yang tersimpan di register Z. Isi register Z ditambah 1 sehingga menjadi $03.
7. Register 18 diisi data ($03) yang tersimpan di alamat RAM yang tersimpan di register Z.
8. Isi dari register 18 ($03) dikeluarkan ke port C.

```
; Program uji instruksi STD dan LDD dengan register Y
; STD Y+q,Rr , LD Rd,Y+q
ldi r28,$02      ; register Y diisi data $02
ldi r16,$88      ; register 16 diisi data $88
std Y+2,r16      ; simpan isi register 16 ke alamat yang ditunjuk register Y ($04)
ldd r17,Y+2      ; ambil dari alamat yang ditunjuk register Y ($04) ke register 17
out $15,r17      ; isi register 17 dikeluarkan ke port C
```

Hasil simulasi:

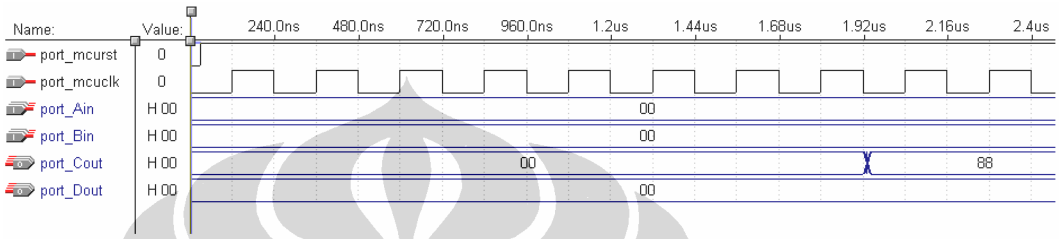


Mikrokontroler diinisialisasi dengan memberikan logika '0' pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian tersebut adalah sebagai berikut:

1. Register 28 atau pointer Y diisi data $02.
2. Register 16 diisi data $88.
3. Instruksi STD akan menyimpan isi register 16 ($88) ke alamat RAM yang merupakan isi dari pointer Y ditambah 2, yaitu $04.
4. Instruksi LDD akan mengambil isi RAM dengan alamat merupakan isi dari pointer Y ditambah 2, yaitu $04, dan disimpan ke register 17.
5. Isi dari register 17 ($88) dikeluarkan ke port C.

```
; Program uji instruksi STD dan LDD dengan register Z
; STD Z+q,Rr , LD Rd,Z+q
ldi r30,$02      ; register Z diisi data $02
ldi r16,$88      ; register 16 diisi data $88
std Z+2,r16      ; simpan isi register 16 ke alamat yang ditunjuk register Z ($04)
ldd r17,Z +2     ; ambil dari alamat yang ditunjuk register Z ($04) ke register 17
out $15,r17      ; isi register 17 dikeluarkan ke port C
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika '0' pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian tersebut adalah sebagai berikut:

1. Register 30 atau pointer Z diisi data $02.
2. Register 16 diisi data $88.
3. Instruksi STD akan menyimpan isi register 16 ($88) ke alamat RAM yang merupakan isi dari pointer Z ditambah 2, yaitu $04.
4. Instruksi LDD akan mengambil isi RAM dengan alamat merupakan isi dari pointer Z ditambah 2, yaitu $04, dan disimpan ke register 17.
5. Isi dari register 17 ($88) dikeluarkan ke port C.

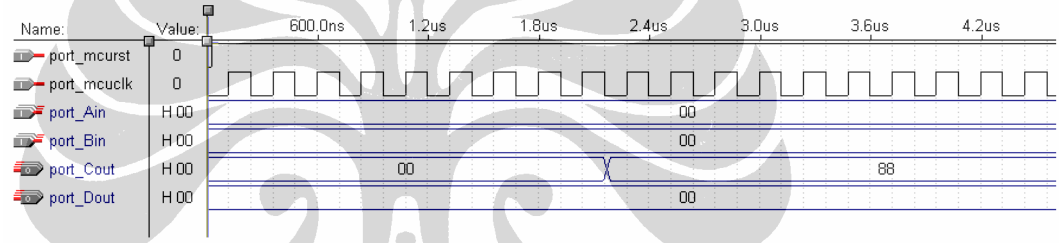
45. PENGUJIAN INSTRUKSI STS

Instruksi STS (*Store Direct to SRAM*) mempunyai bentuk *STS k,Rr*. Instruksi ini akan menyimpan langsung isi register Rr ke alamat RAM k (*(k)ß Rr*).

Pengujian instruksi IN dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi STS
ldi r16,$88     ; register 16 diisi data $88
sts $02,r16     ; isi register 16 disimpan ke alamat RAM $02
ldi r26,$02     ; register 26 diisi data $02
ld r18,X        ; isi alamat RAM yang ditunjuk register X disimpan ke register 18
out $15,r18     ; isi register 18 dikeluarkan ke port C
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian tersebut adalah sebagai berikut:

1. Register 16 diisi data $88.
2. Instruksi STS akan langsung menyimpan isi register 16 ke alamat RAM $02.
3. Register 26 diisi data $02.
4. Instruksi LD akan mengambil isi dari alamat RAM, yaitu $02, yang tersimpan pada register X dan disimpan ke register 18.
5. Isi register 18 dikeluarkan ke port C.

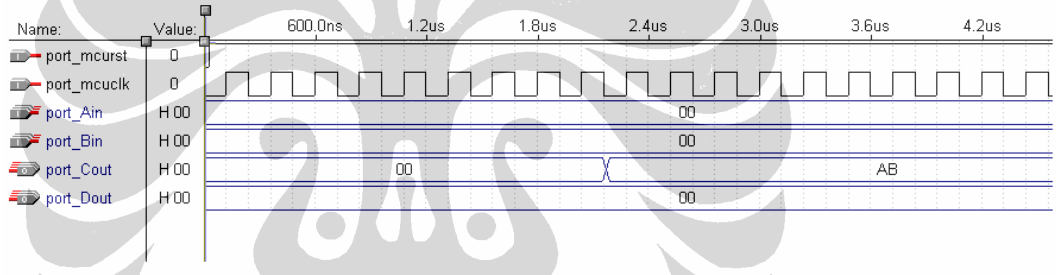
46. PENGUJIAN INSTRUKSI LDS

Instruksi LDS (*Load Direct to SRAM*) mempunyai bentuk *LDS Rd,k*. Instruksi ini akan mengambil langsung isi SRAM dengan alamat *k* dan disimpan di register Rd.

Pengujian instruksi IN dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi LDS
ldi r26,$02     ; register 26 diisi data $02
ldi r16,$AB     ; register 16 diisi data $AB
st X,r16        ; isi register 16 disimpan ke alamat RAM $02
lds r16,$02     ; isi alamat RAM $02 disimpan ke register 16
out $15,r16     ; isi register 16 dikeluarkan ke port C
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika '0' pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian tersebut adalah sebagai berikut:

1. Register 26 diisi data $02.
2. Register 16 diisi data $AB.
3. Instruksi ST akan menyimpan isi register 16 ke alamat RAM yang ditunjuk oleh isi register X.
4. Instruksi LDS akan mengambil isi RAM, yaitu $AB, dengan alamat $02 dan disimpan ke register 16.
5. Isi register 16 dikeluarkan ke port C.

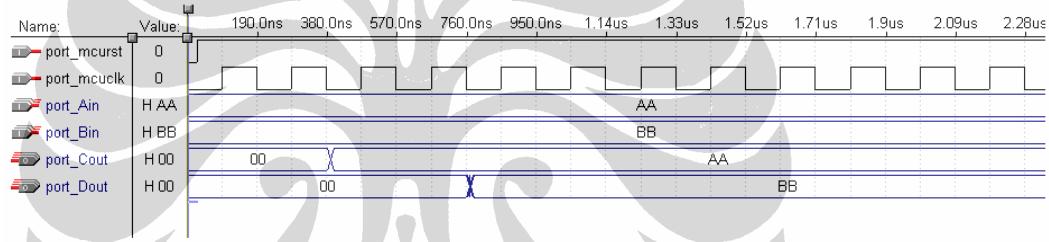
47. PENGUJIAN INSTRUKSI IN

Instruksi IN (*Load an I/O Location to Register*) mempunyai bentuk *IN Rd,A.* Instruksi ini akan mengambil data dari I/O *space*, seperti port register, register konfiigurasi, dan sebagainya, ke dalam register Rd (*Rd ß I/O(A)*).

Pengujian instruksi IN dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi IN
in r16,$1B      ; ambil isi I/O register dengan alamat $1B ke register 16
out $15,r16     ; isi register 16 dikeluarkan ke port C
in r17,$18      ; ambil isi I/O register dengan alamat $18 ke register 17
out $12,r17     ; isi register 17 dikeluarkan ke port D
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi SLEEP, BREAK, RETI adalah sebagai berikut:

1. Isi dari I/O register dengan alamat $1B yaitu hasil pembacaan port A $AA, disimpan ke register 16.
2. Isi dari register 16 dikeluarkan ke port C.
3. Isi dari I/O register dengan alamat $18 yaitu hasil pembacaan port B $BB, disimpan ke register 17.
4. Isi dari register 17 dikeluarkan ke port C.

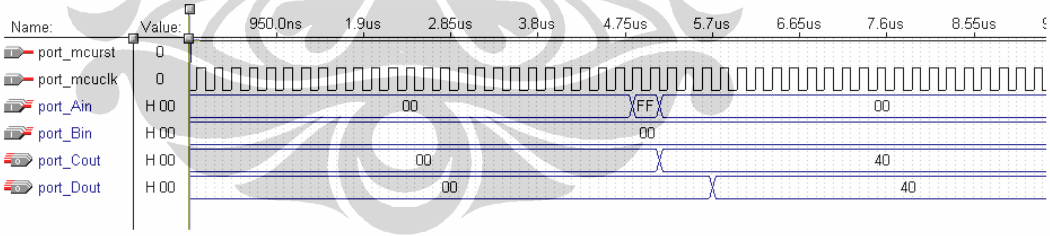
48. PENGUJIAN INSTRUKSI SLEEP, BREAK, RETI

Pada perancangan ini instruksi SLEEP dan BREAK akan menempatkan mikrokontroler berhenti mengeksekusi instruksi. Mikrokontroler akan kembali bekerja bila terjadi *interrupt*. Saat terjadi *interrupt*, mikrokontroler akan mengeksekusi rutin *interrupt*, setelah selesai maka instruksi RETI akan membuat mikrokontroler kembali mengeksekusi instruksi yang ada setelah instruksi SLEEP atau BREAK.

Pengujian instruksi SLEEP, BREAK, dan RETI dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi SLEEP,BREAK,RETI
rjmp reset      ; lompat ke rutin reset
out $15,r17     ; bila terjadi interrupt output port C = $40
reti            ; kembali dari rutin interrupt
reset: ldi r17,$40      ; rutin reset: register 17 diisi $40
out $3B,r17     ; write gicr: enable external interrupt
sei             ; write sreg, flag i='1': enable interrupt
sleep           ; mode sleep
out $12,r17     ; isi register 17 dikeluarkan ke port D
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi SLEEP, BREAK, RETI adalah sebagai berikut:

1. Program pertama kali akan mengarahkan ke rutin Reset, sehingga instruksi yang dieksekusi adalah *ldi r17,$40* yang mengisi register 17 dengan data $40.
2. Instruksi selanjutnya yang dieksekusi adalah *out $3B,r17* yang akan mengirim isi register 17 ke I/O register GICR dengan alamat $3B, sehingga GICR akan

berisi $40 atau B 0100 0000. Posisi bit 6 bernilai '1' yang berarti INT0 (*external interrupt*) di-*enable*.

3. Instruksi SEI akan mengeset bit 7 dari *status register* untuk meng-*enable*-kan *Global Interrupt*.
4. Instruksi SLEEP akan menghentikan operasi dari mikrokontroler hingga terjadi *interrupt*.
5. Saat terjadi *external interrupt*, yaitu port A(7) bernilai '1' maka program akan memanggil alamat rutin *external interrupt* di ROM, yaitu $001. Dengan demikian instruksi yang dieksekusi adalah *out $15,r17* yang akan mengeluarkan isi register 17, yaitu $40, ke port C.
6. Instruksi RETI akan mengembalikan jalannya program ke instruksi setelah instruksi SLEEP atau BREAK, sehingga instruksi yang akan dieksekusi adalah *out $12,r17* yang akan mengeluarkan isi register 17, yaitu $40, ke port D.

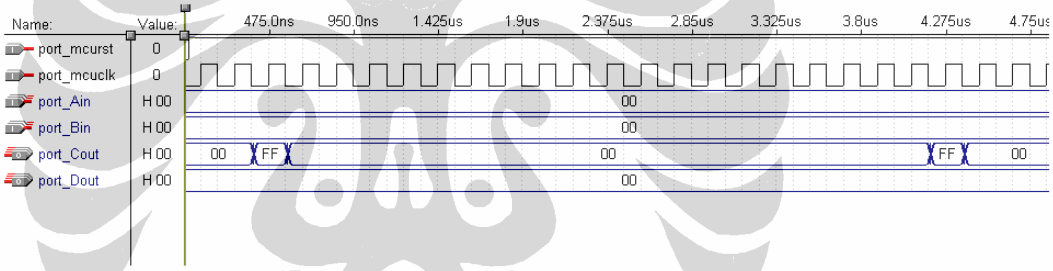
49. PENGUJIAN INSTRUKSI WDR

Instruksi WDR (*Watchdog Reset*) akan me-*reset timer* dari *watchdog timer* dari mikrokontroler, sehingga waktu *reset* akan kembali diulang.

Pengujian instruksi WDR dilakukan dengan membandingkan saat *watchdog timer* bekerja dan instruksi WDR tidak diberikan, dan saat *watchdog timer* bekerja dan instruksi WDR diberikan.

```
; Program uji instruksi WDR
; Watchdog timer bekerja dan instruksi WDR tidak diberikan
reset: ldi r17,$FF      ; register 17 diisi data $FF
out $15,r17             ; isi register 17 dikeluarkan ke port C
out $15,r19             ; isi register 19 dikeluarkan ke port C
ldi r16,$08             ; enable='1',prescaller=000
out $21,r16             ; set I/O register WDTCR
```

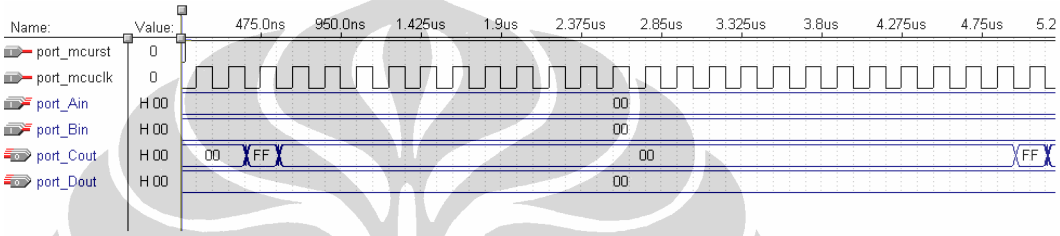
Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Pada keadaan tanpa ada instruksi WDR, maka setelah *watchdog timer* bekerja maka mikrokontroler akan mengalami *reset* dalam waktu tertentu. Namun bila instruksi WDR diberikan maka jangka waktu mikrokontroler untuk mengalami *reset* akan berubah, karena instruksi WDR akan me-*reset watchdog timer*.

```
; Program uji instruksi WDR
; Watchdog timer bekerja dan instruksi WDR tidak diberikan
reset: ldi r17,$FF      ; register 17 diisi data $FF
out $15,r17             ; isi register 17 dikeluarkan ke port C
out $15,r19             ; isi register 19 dikeluarkan ke port C
ldi r16,$08             ; enable='1',prescaller=000
out $21,r16             ; set I/O register WDTCR
nop
nop
wdr                     ; wdr restart
```

Hasil simulasi:



Penggunaan instruksi WDR menyebabkan *watchdog timer* kembali di-*reset* sehingga perhitungan kembali dimulai, dengan demikian *watchdog reset* mengalami penundaan kerja.

50. PENGUJIAN INSTRUKSI LPM

Instruksi LPM (*Load Program Memory*) akan mengambil isi dari ROM sebesar 1 *byte* dengan alamat yang ditunjuk oleh pointer Z dan disimpan ke register Rd. LSB (*Least Significant Bit*) dari isi pointer Z menentukan apakah *low byte* (Z_{LSB}=0) atau *high byte* (Z_{LSB}=1) dari isi ROM yang akan diambil. Instruksi ini mempunyai beberapa bentuk, yaitu:

LPM

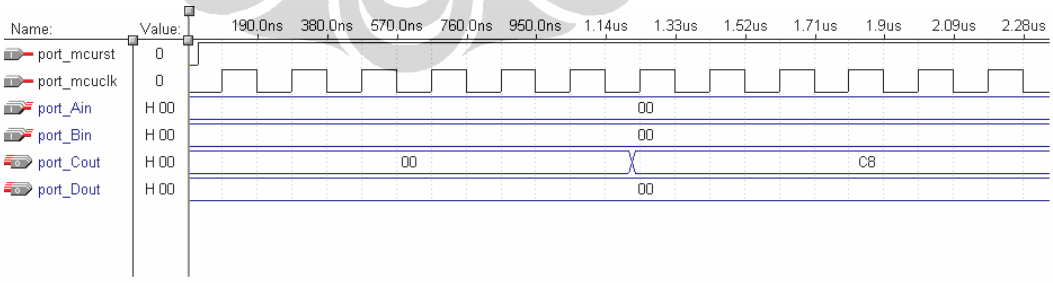
LPM Rd,Z

LPM Rd,Z+

Instruksi LPM akan memakai register 0 sebagai register tujuan, sedangkan bentuk LPM lain tergantung dari Rd. Pengujian instruksi LPM dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program uji instruksi LPM
ldi r30,$02     ; register 30 diisi data $02
                ; alamat: $000, code: $E0E2
ldi r31,$02     ; register 31 diisi data $02
                ; alamat: $001, code: $E0F2
lpm             ; load program memory dengan alamat $0202
                ; alamat: $010, code: $95C8
out $15,r16     ; isi register 16 dikeluarkan ke port C
                ; alamat: $011, code: $BB05
```

Hasil simulasi:



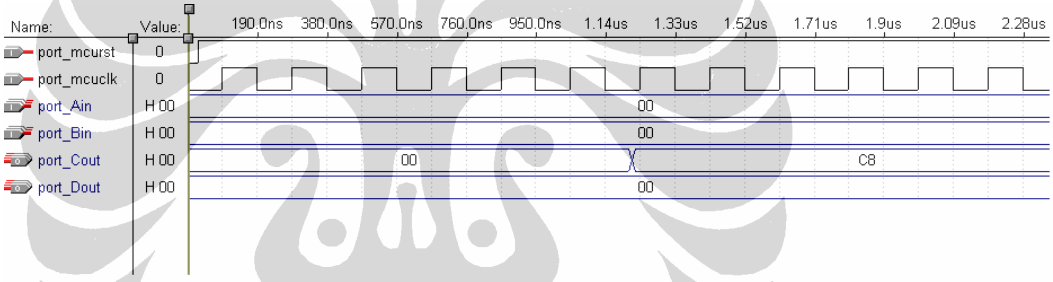
Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi LPM adalah sebagai berikut:

1. Register 30 diisi data $02
2. Register 31 diisi data $02

3. Instruksi LPM akan mengambil isi dari ROM dengan alamat yang tersimpan di register Z, yaitu $0202. Isi dari Z_{LSB} adalah 0 sehingga yang diambil adalah *low byte* dari isi ROM ($C8) dan selanjutnya disimpan ke register 0.

```
; Program uji instruksi LPM Rd,Z
ldi r30,$02     ; register 30 diisi data $02
                ; alamat: $000, code: $E0E2
ldi r31,$02     ; register 31 diisi data $02
                ; alamat: $001, code: $E0F2
lpm r18,Z       ; load program memory dengan alamat $0202
                ; alamat: $010, code: $95C8
out $15,r18     ; isi register 16 dikeluarkan ke port C
                ; alamat: $011, code: $BB05
```

Hasil simulasi:

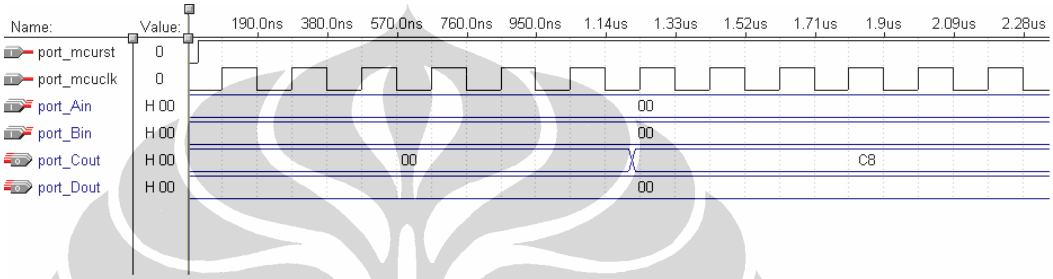


Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi LPM adalah sebagai berikut:

1. Register 30 diisi data $02
2. Register 31 diisi data $02
3. Instruksi LPM akan mengambil isi dari ROM dengan alamat yang tersimpan di register Z, yaitu $0202. Isi dari Z_{LSB} adalah 0 sehingga yang diambil adalah *low byte* dari isi ROM ($C8) dan selanjutnya disimpan ke register 18.

```
; Program uji instruksi LPM Rd,Z+
ldi r30,$02     ; register 30 diisi data $02
                ; alamat: $000, code: $E0E2
ldi r31,$02     ; register 31 diisi data $02
                ; alamat: $001, code: $E0F2
lpm r18,Z+      ; load program memory dengan alamat $0202
                ; alamat: $010, code: $95C8
out $15,r18     ; isi register 16 dikeluarkan ke port C
                ; alamat: $011, code: $BB05
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika '0' pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi LPM adalah sebagai berikut:

1. Register 30 diisi data $02
2. Register 31 diisi data $02
3. Instruksi LPM akan mengambil isi dari ROM dengan alamat yang tersimpan di register Z, yaitu $0202. Isi dari Z_{LSB} adalah 0 sehingga yang diambil adalah *low byte* dari isi ROM ($C8) dan selanjutnya disimpan ke register 18.

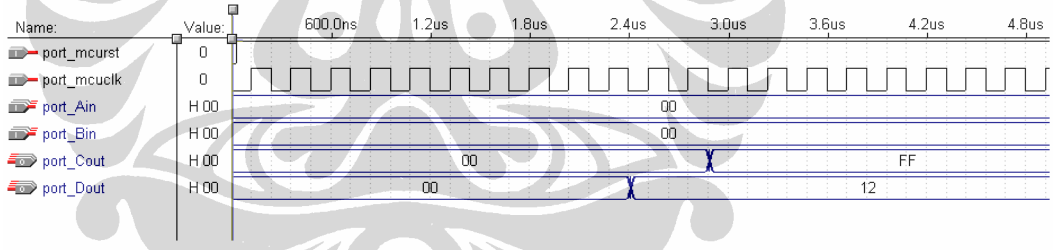
51. PENGUJIAN INSTRUKSI SPM

Instruksi SPM (*Store Program Memory*) dapat digunakan untuk mengganti isi dari *program memory*. Instruksi ini menggunakan register Z sebagai alamat dan pasangan register R1:R0 sebagai sumber data.

Pengujian instruksi LPM dilakukan dengan memberikan program berikut ini pada ROM UIMega 8535 dan mengamati hasil keluarannya:

```
; Program Uji Instruksi SPM
; Data yang akan diisi ke ROM
; BB12 adalah opcode untuk instruksi out $12,r17
ldi r16,$BB     ;isi r0: $BB
ldi r17,$12     ;isi r1: $12
; Alamat ROM yang akan dituju
ldi r30,$05
ldi r31,$00
spm
out $15,r16
ldi r18,$FF
out $15,r18
```

Hasil simulasi:



Mikrokontroler diinisialisasi dengan memberikan logika ‘0’ pada pin *reset* (*port_mcurst*). Proses yang dilakukan oleh pengujian instruksi LPM adalah sebagai berikut:

1. Register 16 diisi dengan data $BB. Register 16 dalam perancangan ini mengacu pada register 0.
2. Register 17 diisi dengan data $12. Register 17 dalam perancangan ini mengacu pada register 1.
3. Register 30 diisi data $05.
4. Register 31 diisi data $00.

5. Keempat proses ini bertujuan untuk mengisi alamat ROM $005 dengan data $BB12 yang mengacu pada opcode untuk perintah *out $12,r17* , yaitu mengeluarkan isi register 17 ($12) ke port D. Penggantian isi pada alamat ROM $005 berarti akan menghapus perintah *out $15,r16*.
6. Register 18 diisi data $FF.
7. Isi register 18 dikeluarkan ke port C.

LAMPIRAN 5

SOURCE CODE VHDL

Source code uimega8535.vhd

```
library ieee;
use ieee.std_logic_1164.all;
entity UIMega8535 is
port(    port_mcuclk,port_mcurst : in std_logic;
         port_Ain,port_Bin : in std_logic_vector(7 downto 0);
         port_Cout,port_Dout : out std_logic_vector(7 downto 0));
end UIMega8535;
architecture Arch_UIMega8535 of UIMega8535 is
component ROMUnit
port(    pin_pc          : in std_logic_vector(8 downto 0);
         pin_instrom     : out std_logic_vector(15 downto 0);
         pin_datain      : in std_logic_vector(15 downto 0);
         pin_werom       : in std_logic);
end component;
component InstructionRegUnit
port(    pin_in : in std_logic_vector(15 downto 0);
         pin_out : out std_logic_vector(15 downto 0));
end component;
component CoreUnit
port(    pin_clk,pin_rst : in std_logic;
         pin_instir : in std_logic_vector(15 downto 0);
         pin_Ainport,pin_Binport : in std_logic_vector(7 downto 0);
         pin_Coutport,pin_Doutport : out std_logic_vector(7 downto 0);
         pin_addrrom : buffer std_logic_vector(8 downto 0);
         pin_tov : in std_logic;
         pin_wrtcnt : out std_logic;
         pin_tcnt : out std_logic_vector(7 downto 0);
         pin_toie : out std_logic;
         pin_cso : out std_logic_vector(2 downto 0);
         pin_wde : out std_logic;
         pin_wdp : out std_logic_vector(2 downto 0);
         pin_wdrrestart : out std_logic);
end component;
component TimerInterruptUnit
port(    pin_clk,pin_rst, pin_toie,pin_wrtcnt : in std_logic;
         pin_tcnt : in std_logic_vector(7 downto 0);
         pin_cso : in std_logic_vector(2 downto 0);
         pin_Binport7 : in std_logic;
         pin_tov : out std_logic);
end component;
component prescallerWDR
port(    pin_clk,pin_rst : in std_logic;
         pin_wde : in std_logic;
         pin_wdp : in std_logic_vector(2 downto 0);
         pin_wdrrst : out std_logic);
end component;
signal s_masterclk,s_masterrst, s_werom,s_tov,s_wrtcnt,s_toie : std_logic;
signal s_pc : std_logic_vector(8 downto 0);
signal s_instrom,s_outrom,s_datainrom : std_logic_vector(15 downto 0);
signal s_tcnt : std_logic_vector(7 downto 0);
signal s_cso,s_wdp : std_logic_vector(2 downto 0);
signal s_extportEX,s_extportTM,s_wde, s_wdrrestart,wdrrestart,s_wdrrst,masterrst : std_logic;
begin
Unit_ROMUnit: ROMUnit port map (s_pc,s_instrom,s_datainrom,s_werom);
Unit_InstructionReg: InstructionRegUnit port map (s_instrom,s_outrom);
Unit_Core:CoreUnit port map(port_mcuclk,masterrst,s_outrom,port_Ain,port_Bin,port_Cout,
port_Dout,s_pc,s_tov,s_wrtcnt,s_tcnt,s_toie,s_cso,s_wde,s_wdp,s_wdrrestart,s_datainrom,
s_werom);
-- port B (7) -> timer irq
s_extportTM <= port_Bin(7);
Unit_TimerIrq : TimerInterruptUnit
port map (port_mcuclk,masterrst,s_toie,s_wrtcnt,s_tcnt,s_cso,s_extportTM,s_tov);
Unit_WDR : prescallerWDR port map (port_mcuclk,wdrrestart,s_wde,s_wdp,s_wdrrst);
wdrrestart <= port_mcurst and s_wdrrestart;
masterrst <= port_mcurst and not s_wdrrst;
end Arch_UIMega8535;
```

Source code ROMUnit.vhd

```
library ieee;
use ieee.std_logic_1164.all;
library lpm;
use lpm.lpm_components.all;
entity ROMUnit IS
PORT(   pin_pc          : in std_logic_vector(8 downto 0);
        pin_instrom     : out std_logic_vector(15 downto 0);
        pin_datain      : in std_logic_vector(15 downto 0);
        pin_werom       : in std_logic);
end ROMUnit;
architecture Arch_ROMUnit of ROMUnit is
        COMPONENT lpm_ram_dq
        GENERIC (LPM_WIDTH              : NATURAL;
                 LPM_WIDTHAD            : NATURAL;
                 LPM_INDATA             : STRING;
                 LPM_ADDRESS_CONTROL    : STRING;
                 LPM_OUTDATA: STRING;
                 LPM_FILE               : STRING;
                 LPM_HINT               : STRING);
        PORT (  address : IN STD_LOGIC_VECTOR (8 DOWNTO 0);
                q       : OUT STD_LOGIC_VECTOR (15 DOWNTO 0);
                data    : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
                we      : IN STD_LOGIC);
        END COMPONENT;
begin
        lpm_ram_dq_component : lpm_ram_dq
        GENERIC MAP (
                LPM_WIDTH => 16,
                LPM_WIDTHAD => 9,
                LPM_INDATA => "UNREGISTERED",
                LPM_ADDRESS_CONTROL => "UNREGISTERED",
                LPM_OUTDATA => "UNREGISTERED",
                LPM_FILE => "program.hex",
                LPM_HINT => "USE_EAB=ON")
        PORT MAP (address => pin_pc,data => pin_datain,we => pin_werom,
                  q => pin_instrom);
end Arch_ROMUnit;
```

Source code InstructionRegUnit.vhd

```
library ieee;
use ieee.std_logic_1164.all;
entity InstructionRegUnit is
port(pin_in : in std_logic_vector(15 downto 0);
     pin_out : out std_logic_vector(15 downto 0));
end InstructionRegUnit;
architecture Arch_InstructionRegUnit of InstructionRegUnit is
begin
process(pin_in)
begin
        pin_out(15 downto 8) <= pin_in(7 downto 0);
        pin_out(7 downto 0) <= pin_in(15 downto 8);
end process;
end Arch_InstructionRegUnit;
```

Source code CoreUnit.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
library lpm;
use lpm.lpm_components.all;
entity CoreUnit is
port(   pin_clk,pin_rst : in std_logic;
        pin_instir : in std_logic_vector(15 downto 0);
```

```
        pin_Ainport,pin_Binport : in std_logic_vector(7 downto 0);          -- A in, B in
        pin_Coutport,pin_Doutport : out std_logic_vector(7 downto 0);       -- C out, D out
        pin_addrrom : buffer std_logic_vector(8 downto 0);
        pin_tov : in std_logic;
        pin_wrtcnt : out std_logic;
        pin_tcnt : out std_logic_vector(7 downto 0);
        pin_toie : out std_logic;
        pin_cso : out std_logic_vector(2 downto 0);
        pin_wde : out std_logic;
        pin_wdp : out std_logic_vector(2 downto 0);
        pin_wdrrestart : out std_logic;
        pin_wrdatarom : out std_logic_vector(15 downto 0);
        pin_werom : out std_logic );
end CoreUnit;
architecture Arch_CoreUnit of CoreUnit is
type statetype is (exestate,standbystate,exestate32,store2,store3,store4,getaddress,store2sts,
        lpmstate,cpsestate,sbrcsstate,sbicsstate,spmstore1,spmstore2,spmstore3,spmstore4);
signal state : statetype;
signal s_timsk,s_tccr : std_logic_vector(7 downto 0);
signal s_gprwrite, s_gprwritelow : std_logic;
signal s_cp,s_cpc,s_sbc,s_addlsl,s_cpse,s_sub,s_adcrol,s_andtst,s_eorclr,s_or,s_mov,s_cpi,
        s_sbci,s_subi,s_orisbr,s_andicbr,s_ldx,s_ldxinc,s_lddecx,s_ldy,s_ldyinc,s_lddecy,
        s_ldz,s_ldzinc,s_lddecz,s_stx,s_stxinc,s_stdecx,s_sty,s_styinc,s_stdecy,
        s_stz,s_stzinc,s_stdecz,s_com,s_neg,s_swap,s_inc,s_asr,s_lsr,s_ror,s_dec,s_bset,
        s_bclr,s_ret,s_reti,s_sleep,s_sbicbi,s_sbics,s_in,s_out,s_rjmp,s_rcall,s_serldi,
        s_brcsbc,s_bld,s_bst,s_sbrcs,s_push,s_pop,s_break,s_wdr : std_logic;
signal s_movw,s_muls,s_mulsu,s_fmul,s_fmuls,s_fmulsu,s_lddzq,s_lddyq,s_stdzq,
        s_stdyq,s_lds,s_lpmz,s_lpmzinc,s_sts,s_lpm,s_spm,s_ijmp,s_icall,s_adiw,
        s_sbiw,s_mul : std_logic;
signal opra,oprb,oprab, s_bldout : std_logic_vector(7 downto 0);
signal s_bstout : std_logic;
signal s_regd32inst : std_logic_vector(3 downto 0);
signal s_stat32,s_statst,s_statst2,s_statst3,s_statget,s_statstore,s_statlpm,s_statcpse,
        s_statsbrcs,s_statsbics : std_logic;
signal s_statspm0,s_statspm,s_statspm1,s_statspm2 : std_logic;
signal is_sbrc,is_sbrs,is_sbis,is_sbic : std_logic;
signal bit_b : std_logic_vector(2 downto 0);
component PCUnit
port(   pin_clk,pin_rst : in std_logic;
        pin_enable : in std_logic;
        pin_pc : buffer std_logic_vector(8 downto 0);
        pin_addr : in std_logic_vector(8 downto 0);
        pin_indirect : in std_logic;
        pin_int0int,pin_timerint : in std_logic);
end component;
signal s_enablepc, s_indirect, s_int0int, s_timerint, logicsig : std_logic;
signal s_addresspc : std_logic_vector(8 downto 0);
signal logicsel : integer range 0 to 3; --(0:and,andi,1:or,ori,2:eor,3:com)
signal s_logicout : std_logic_vector(7 downto 0);
signal shiftsig, arithsig, multiplysig : std_logic;
signal shiftsel : integer range 0 to 2; --(0:lsr,1:ror,2:asr)
signal s_shiftout : std_logic_vector(7 downto 0);
signal shifter_enable,shifter_cflagin,shifter_cflagout : std_logic;
signal shifter_in,shifter_out : std_logic_vector(7 downto 0);
signal shifter_selop : integer range 0 to 2;
component ShifterUnit
port(   pin_enable : in std_logic;
  pin_in : in std_logic_vector(7 downto 0);
  pin_selop : in integer range 0 to 2;
  pin_cflagin : in std_logic;
  pin_cflagout : out std_logic;
  pin_out : out std_logic_vector(7 downto 0));
end component;
signal logic_enable : std_logic;
signal logic_opra,logic_oprb,logic_out : std_logic_vector(7 downto 0);
signal logic_selop : integer range 0 to 3;
component LogicUnit
port(   pin_enable : in std_logic;
  pin_opra,pin_oprb : in std_logic_vector(7 downto 0);
```

```
   pin_selop : in integer range 0 to 3;
   pin_out : out std_logic_vector(7 downto 0));
end component;
signal add_cflagin, addsig, add_cout,add_overflow : std_logic;
signal add_out,s_arithout,swap_in,swap_out, s_swapout : std_logic_vector(7 downto 0);
component SwapUnit
port(     pin_in : in std_logic_vector(7 downto 0);
          pin_out : out std_logic_vector(7 downto 0));
end component;
signal mult_a,mult_b : std_logic_vector(7 downto 0);
signal mult_sign, s_sign, mult_shift, s_shift : std_logic;
signal mult_resulthigh,mult_resultlow : std_logic_vector (7 downto 0);
signal s_multouthigh,s_multoutlow : std_logic_vector (7 downto 0);
signal mult_carry,s_multcarry : std_logic;
signal multsu_resulthigh,multsu_resultlow : std_logic_vector (7 downto 0);
signal s_multsuouthigh,s_multsuoutlow : std_logic_vector (7 downto 0);
signal multsu_carry,s_multsucarry : std_logic;

component MultUnit
port(     a: in std_logic_vector (7 downto 0);
          b: in std_logic_vector (7 downto 0);
          sign : in std_logic;
          shift : in std_logic;
          result_high: out std_logic_vector (7 downto 0);
          result_low: out std_logic_vector (7 downto 0);
          pin_carry : out std_logic);
end component;
component MulsuUnit
port(     a,b : in std_logic_vector(7 downto 0);
          shift : in std_logic;
          result_high: out std_logic_vector (7 downto 0);
          result_low: out std_logic_vector (7 downto 0);
          pin_carry : out std_logic);
end component;

signal s_addsubiwout1,s_addsubiwout2 : std_logic_vector(7 downto 0);
signal AddLowHighCin, AddLowHighAddSub : std_logic;
signal s_AddLowCout,s_AddHighCout,s_AddLowOv,s_AddHighOv : std_logic;

component AddLowUnit
port(     pin_opra,pin_oprb : in std_logic_vector(7 downto 0);
          pin_cin,pin_addsub : in std_logic;
          pin_result : out std_logic_vector(7 downto 0);
          pin_cout,pin_overflow : out std_logic);
end component;

component AddHighUnit
port(     pin_opra,pin_oprb : in std_logic_vector(7 downto 0);
          pin_cin,pin_addsub : in std_logic;
          pin_result : out std_logic_vector(7 downto 0);
          pin_cout,pin_overflow : out std_logic);
end component;

signal HFlag,SFlag,VFlag,NFlag,ZFlag,CFlag : std_logic;
signal s_addrport : std_logic_vector(5 downto 0);
type gprtype is array (0 to 15) of std_logic_vector(7 downto 0);
signal gpr : gprtype;
signal s_zhigh, s_dupx,s_dupy,s_dupz, rd,rr : std_logic_vector(7 downto 0);
signal sr : std_logic_vector(7 downto 0);
signal stack0,stack1,stack2,stack3,stack4,stack5,stack6,stack7 : std_logic_vector(8 downto 0);
signal tpstack : std_logic_vector(8 downto 0);
signal s_instir : std_logic_vector(7 downto 0);
signal s_zinclpm : std_logic;
signal s_gicr : std_logic_vector(7 downto 0);
component buffExtInt
port(     pin_enable : in std_logic;
          pin_in     : in std_logic;
          pin_out    : out std_logic);
end component;
```

```
signal ena_extirq,Ainport7 : std_logic;
signal s_wdtcr : std_logic_vector(7 downto 0);
component RAMUnit
port(     address          : in std_logic_vector(7 downto 0);
          we               : in std_logic;
          data             : in std_logic_vector(7 downto 0);
          q                : out std_logic_vector(7 downto 0));
end component;
signal s_addressram_dup,s_datainram_dup : std_logic_vector(7 downto 0);
signal    s_addressram, s_datainram, s_outram : std_logic_vector(7 downto 0);
signal    s_enastclk : std_logic;
begin
Decodeprocess: process(pin_instir,s_stat32,s_statst,s_statst2,s_statst3,s_statget,s_statstore,
s_statlpm,s_statcpse,s_statsbrcs,s_statsbics,s_statspm0,s_statspm,s_statspm1,s_statspm2)
begin
s_cpc<='0';s_sbc<='0';s_addlsl<='0';s_cpse<='0';s_sub<='0';s_adcrol<='0';s_andtst<='0';
s_eorclr<='0';s_or <='0';s_mov<='0';s_cpi<='0';s_sbci<='0';s_subi<='0';s_orisbr<='0';
s_andicbr<='0';s_ldx<='0';s_ldxinc<='0';s_lddecx<='0';s_ldy<='0';s_ldyinc<='0';s_lddecy<='0';
s_ldz <= '0';s_ldzinc <= '0';s_lddecz <= '0';s_stx <= '0';s_stxinc <= '0';s_stdecx <= '0';
s_sty <= '0';s_styinc <= '0';s_stdecy <= '0';s_stz <= '0';s_stzinc <= '0';s_stdecz <= '0';
s_com <= '0';s_neg <= '0';s_swap <= '0';s_inc <= '0';s_asr <= '0';s_lsr <= '0';
s_ror <= '0';s_dec <= '0';s_bset <= '0';s_bclr <= '0';s_ret <= '0';s_reti <= '0';
s_sleep <= '0';s_sbicbi <= '0';s_sbics <= '0';s_in <= '0';s_out <= '0';s_rjmp <= '0';
s_rcall<='0';s_serldi<='0';s_brcsbc<='0';s_bld <= '0';s_bst <= '0';s_sbrcs <= '0';s_push <= '0';
s_pop <= '0';s_movw <= '0';s_muls <= '0';s_mulsu <= '0';s_fmul <= '0';s_fmuls <= '0';
s_fmulsu  <=  '0';s_lddzq<='0';s_lddyq<='0';s_stdzq<='0';s_stdyq<='0';s_lds<='0';s_lpmz<='0';
s_lpmzinc<='0';s_sts<='0';s_lpm<='0';s_spm <= '0';s_ijmp <= '0';s_icall <= '0';s_adiw <= '0';
s_sbiw <= '0';s_mul <= '0'; s_break <= '0';s_cp <= '0'; s_wdr <= '0';
logicsig <= '0';shiftsig <= '0';arithsig <= '0';multiplysig <= '0';s_shift <= '0';addsig <= '0';
add_cflagin <= '0';AddLowHighCin <= '0';AddLowHighAddSub <= '0';
s_bldout <= "00000000";s_bstout <= '0';s_indirect <= '0';s_zinclpm <= '0';
is_sbrc <= '0';is_sbrs <= '0';is_sbic <= '0';is_sbis <= '0';

if (s_stat32='0' and s_statst='0'  and s_statst2='0' and s_statst3='0' and s_statget='0' and
s_statstore='0' and s_statlpm='0' and s_statcpse='0' and s_statsbrcs='0' and s_statsbics='0'
and s_statspm0='0' and s_statspm='0' and s_statspm1='0' and s_statspm2='0') then

s_addressram <= "00000000";s_datainram <= "00000000";
case pin_instir(15 downto 12) is
          when "0000" =>
          s_addresspc <= "000000000";
                    if pin_instir(11 downto 10)="00" then
                              if pin_instir(9 downto 8)="00" then
                                        -- NOP
                              elsif pin_instir(9 downto 8)="01" then
                                        -- MOVW
                                        s_movw <= '1';
                                        opra <= gpr(2);oprb <= gpr(3);
                              elsif pin_instir(9 downto 8)="10" then
                                        -- MULS
                                        s_muls <= '1';multiplysig <= '1';s_sign <= '1';s_shift <= '0';
                                        opra <= gpr(conv_integer(pin_instir(7 downto 4)));
                                        oprb <= gpr(conv_integer(pin_instir(3 downto 0)));
                              elsif pin_instir(9 downto 8)="11" then
                                        if pin_instir(7)='0' then
                                                  if pin_instir(3)='0' then
                                                            -- MULSU
                                                  s_mulsu <= '1';multiplysig <= '1';s_shift <= '0';
                                                  opra <= gpr(conv_integer(pin_instir(6 downto 4)));
                                                  oprb <= gpr(conv_integer(pin_instir(2 downto 0)));
                                                  elsif pin_instir(3)='1' then
                                                            -- FMUL
                                                  s_fmul <= '1';multiplysig <= '1';s_shift <= '1';
                                                            s_sign <= '0';
                                                  opra <= gpr(conv_integer(pin_instir(6 downto 4)));
                                                  oprb <= gpr(conv_integer(pin_instir(2 downto 0)));
                                                  end if;
                                        elsif pin_instir(7)='1' then
                                                  if pin_instir(3)='0' then
```

```
                                        -- FMULS
                                    s_fmuls <= '1';multiplysig <= '1';s_shift <= '1';
                                    s_sign <= '1';
                                    opra <= gpr(conv_integer(pin_instir(6 downto 4)));
                                    oprb <= gpr(conv_integer(pin_instir(2 downto 0)));
                                    elsif pin_instir(3)='1' then
                                        -- FMULSU
                                    s_fmulsu <= '1';multiplysig <= '1';s_shift <= '1';
                                    opra <= gpr(conv_integer(pin_instir(6 downto 4)));
                                    oprb <= gpr(conv_integer(pin_instir(2 downto 0)));
                                    end if;
                            end if;
                    end if;
            elsif pin_instir(11 downto 10)="01" then
                    -- CPC
                    s_cpc <= '1';addsig <= '0';add_cflagin <= not sr(0);
                    opra <= gpr(conv_integer(pin_instir(7 downto 4)));
                    oprb <= gpr(conv_integer(pin_instir(3 downto 0)));
            elsif pin_instir(11 downto 10)="10" then
                    -- SBC
                    s_sbc <= '1';arithsig <= '1';addsig <= '0';add_cflagin <= not sr(0);
                    opra <= gpr(conv_integer(pin_instir(7 downto 4)));
                    oprb <= gpr(conv_integer(pin_instir(3 downto 0)));
            elsif pin_instir(11 downto 10)="11" then
                    -- ADD,LSL
                    s_addlsl <= '1';arithsig <= '1';addsig <= '1';add_cflagin <= '0';
                    opra <= gpr(conv_integer(pin_instir(7 downto 4)));
                    oprb <= gpr(conv_integer(pin_instir(3 downto 0)));
            end if;
    when "0001" =>
            if pin_instir(11 downto 10)="00" then
                    -- CPSE
                    s_cpse <= '1'; tpstack <= pin_addrrom + 1;
                    rd <= gpr(conv_integer(pin_instir(7 downto 4)));
                    rr <= gpr(conv_integer(pin_instir(3 downto 0)));
            elsif pin_instir(11 downto 10)="01" then
                    -- CP
                    s_cp <= '1';s_addresspc <= "000000000";addsig <= '0';
                    add_cflagin <= '1';opra <= gpr(conv_integer(pin_instir(7 downto 4)));
                    oprb <= gpr(conv_integer(pin_instir(3 downto 0)));
            elsif pin_instir(11 downto 10)="10" then
                    -- SUB
                    s_sub <= '1';s_addresspc <= "000000000";arithsig <= '1';
                    addsig <= '0';add_cflagin <= '1';
                    opra <= gpr(conv_integer(pin_instir(7 downto 4)));
                    oprb <= gpr(conv_integer(pin_instir(3 downto 0)));
            elsif pin_instir(11 downto 10)="11" then
                    -- ADC,ROL
                    s_adcrol <= '1';s_addresspc <= "000000000";arithsig <= '1';
                    addsig <= '1';add_cflagin <= sr(0);
                    opra <= gpr(conv_integer(pin_instir(7 downto 4)));
                    oprb <= gpr(conv_integer(pin_instir(3 downto 0)));
            end if;
    when "0010" =>
            s_addresspc <= "000000000";
            if pin_instir(11 downto 10)="00" then
                    -- AND,TST
                    s_andtst <= '1';logicsel <= 0;logicsig <= '1';
                    opra <= gpr(conv_integer(pin_instir(7 downto 4)));
                    oprb <= gpr(conv_integer(pin_instir(3 downto 0)));
            elsif pin_instir(11 downto 10)="01" then
                    -- EOR,CLR
                    s_eorclr <= '1';logicsel <= 2;logicsig <= '1';
                    opra <= gpr(conv_integer(pin_instir(7 downto 4)));
                    oprb <= gpr(conv_integer(pin_instir(3 downto 0)));
            elsif pin_instir(11 downto 10)="10" then
                    -- OR
                    s_or <= '1';logicsel <= 1; logicsig <= '1';
                    opra <= gpr(conv_integer(pin_instir(7 downto 4)));
```

```
                        oprb <= gpr(conv_integer(pin_instir(3 downto 0)));
                elsif pin_instir(11 downto 10)="11" then
                        -- MOV
                        s_mov <= '1';
                end if;
        when "0011" =>
                s_addresspc <= "000000000";
                -- CPI
                s_cpi <= '1';addsig <= '0';add_cflagin <= '1';
                opra <= gpr(conv_integer(pin_instir(7 downto 4)));
                oprb <= pin_instir(11 downto 8) & pin_instir(3 downto 0);
        when "0100" =>
                s_addresspc <= "000000000";
                -- SBCI
                s_sbci <= '1';arithsig <= '1';addsig <= '0';add_cflagin <= not sr(0);
                opra <= gpr(conv_integer(pin_instir(7 downto 4)));
                oprb <= pin_instir(11 downto 8) & pin_instir(3 downto 0);
        when "0101" =>
                s_addresspc <= "000000000";
                -- SUBI
                s_subi <= '1';arithsig <= '1';addsig <= '0';add_cflagin <= '1';
                opra <= gpr(conv_integer(pin_instir(7 downto 4)));
                oprb <= pin_instir(11 downto 8) & pin_instir(3 downto 0);
        when "0110" =>
                s_addresspc <= "000000000";
                -- ORI,SBR
                s_orisbr <= '1';logicsel <= 1;logicsig <= '1';
                opra <= gpr(conv_integer(pin_instir(7 downto 4)));
                oprb <= pin_instir(11 downto 8) & pin_instir(3 downto 0);
        when "0111" =>
                s_addresspc <= "000000000";
                -- ANDI,CBR
                s_andicbr <= '1'; logicsel <= 0;logicsig <= '1';
                opra <= gpr(conv_integer(pin_instir(7 downto 4)));
                oprb <= pin_instir(11 downto 8) & pin_instir(3 downto 0);
        when "1000" =>
                s_addresspc <= "000000000";
                if pin_instir(9)='0' then
                        if pin_instir(3)='0' then
                                if pin_instir(2 downto 0)="000" then
                                        -- LD(Rd,Z)
                                        s_ldz <= '1';s_addresspc <= "000000000";
                                        s_addressram <= s_dupz;
                                else
                                        -- LDD(Rd,Z+q)
                                        s_lddzq <= '1';s_addresspc <= "000000000";
s_addressram <= s_dupz + ("00" & pin_instir(13) & pin_instir(11 downto 10) & pin_instir(2
downto 0));
                                end if;
                        elsif pin_instir(3)='1' then
                                if pin_instir(2 downto 0)="000" then
                                        -- LD(Rd,Y)
                                        s_ldy <= '1';s_addresspc <= "000000000";
                                        s_addressram <= s_dupy;
                                else
                                        -- LDD(Rd,Y+q)
                                        s_lddyq <= '1';s_addresspc <= "000000000";
s_addressram <= s_dupy + ("00" & pin_instir(13) & pin_instir(11 downto 10) & pin_instir(2
downto 0));
                                end if;
                        end if;
                elsif pin_instir(9)='1' then
                        if pin_instir(3)='0' then
                                if pin_instir(2 downto 0)="000" then
                                        -- ST(Z,Rr)
                                        s_stz <= '1';tpstack <= pin_addrrom + 1;
                                s_addressram <= s_dupz;s_addressram_dup <= s_dupz;
                                s_datainram <= gpr(conv_integer(pin_instir(7 downto 4)));
                        s_datainram_dup <= gpr(conv_integer(pin_instir(7 downto 4)));
```

```
                                else
                                        -- STD(Z+q,Rr)
                                        s_stdzq <= '1';tpstack <= pin_addrrom + 1;
s_addressram <= s_dupz + ("00" & pin_instir(13) & pin_instir(11 downto 10) & pin_instir(2
downto 0));
s_addressram_dup <= s_dupz + ("00" & pin_instir(13) & pin_instir(11 downto 10) & pin_instir(2
downto 0));
        s_datainram <= gpr(conv_integer(pin_instir(7 downto 4)));
        s_datainram_dup <= gpr(conv_integer(pin_instir(7 downto 4)));
                                end if;
                        elsif pin_instir(3)='1' then
                                if pin_instir(2 downto 0)="000" then
                                        -- ST(Y,Rr)
                                        s_sty <= '1';tpstack <= pin_addrrom + 1;
                                s_addressram <= s_dupy;s_addressram_dup <= s_dupy;
                                s_datainram <= gpr(conv_integer(pin_instir(7 downto 4)));
                        s_datainram_dup <= gpr(conv_integer(pin_instir(7 downto 4)));
                                else
                                        -- STD(Y+q,Rr)
                                        s_stdyq <= '1';tpstack <= pin_addrrom + 1;
s_addressram <= s_dupy + ("00" & pin_instir(13) & pin_instir(11 downto 10) & pin_instir(2
downto 0));
s_addressram_dup <= s_dupy + ("00" & pin_instir(13) & pin_instir(11 downto 10) & pin_instir(2
downto 0));
                                s_datainram <= gpr(conv_integer(pin_instir(7 downto 4)));
                        s_datainram_dup <= gpr(conv_integer(pin_instir(7 downto 4)));
                        end if;
                        end if;
                end if;
        when "1001" =>
                if pin_instir(11 downto 9)="000" then
                        if pin_instir(3 downto 0)="0000" then
                                -- LDS
                                s_lds <= '1';s_addresspc <= "000000000";
                                s_regd32inst <= pin_instir(7 downto 4);
                        elsif pin_instir(3 downto 0)="0001" then
                                -- LD(Rd,Z+)
                                s_ldzinc <= '1';s_addresspc <= "000000000";
                                s_addressram <= s_dupz;
                        elsif pin_instir(3 downto 0)="0010" then
                                -- LD(Rd,-Z)
                                s_lddecz <= '1';s_addresspc <= "000000000";
                                s_addressram <= s_dupz - 1;
                        elsif pin_instir(3 downto 0)="0100" then
                                -- LPM(Rd,Z)
                                s_lpmz <= '1';s_indirect <= '1';
                                s_addresspc <= s_zhigh(0) & s_dupz;
                                s_regd32inst <= pin_instir(7 downto 4);
                                tpstack <= pin_addrrom + 1;
                        elsif pin_instir(3 downto 0)="0101" then
                                -- LPM(Rd,Z+)
                                s_lpmzinc <= '1';s_indirect <= '1';
                                s_addresspc <= s_zhigh(0) & s_dupz;
                                s_regd32inst <= pin_instir(7 downto 4);
                                tpstack <= pin_addrrom + 1;
                                s_zinclpm <= '1';
                        elsif pin_instir(3 downto 0)="1001" then
                                -- LD(Rd,Y+)
                                s_ldyinc <= '1';s_addresspc <= "000000000";
                                s_addressram <= s_dupy;
                        elsif pin_instir(3 downto 0)="1010" then
                                -- LD(Rd,-Y)
                                s_lddecy <= '1';s_addresspc <= "000000000";
                                s_addressram <= s_dupy - 1;
                        elsif pin_instir(3 downto 0)="1100" then
                                -- LD(Rd,X)
                                s_ldx <= '1';s_addresspc <= "000000000";
                                s_addressram <= s_dupx;
                        elsif pin_instir(3 downto 0)="1101" then
```

```
                -- LD(Rd,X+)
                s_ldxinc <= '1';s_addresspc <= "000000000";
                s_addressram <= s_dupx;
        elsif pin_instir(3 downto 0)="1110" then
                -- LD(Rd,-X)
                s_lddecx <= '1';s_addresspc <= "000000000";
                s_addressram <= s_dupx - 1;
        elsif pin_instir(3 downto 0)="1111" then
                -- POP
                s_pop <= '1';s_addresspc <= "000000000";
        end if;
elsif pin_instir(11 downto 9)="001" then
        if pin_instir(3 downto 0)="0000" then
                -- STS
                s_sts <= '1';s_addresspc <= "000000000";
                s_regd32inst <= pin_instir(7 downto 4);
        elsif pin_instir(3 downto 0)="0001" then
                -- ST(Z+,Rr)
                s_stzinc <= '1';tpstack <= pin_addrrom + 1;
                s_addressram <= s_dupz;s_addressram_dup <= s_dupz;
                s_datainram <= gpr(conv_integer(pin_instir(7 downto 4)));
        s_datainram_dup <= gpr(conv_integer(pin_instir(7 downto 4)));
        elsif pin_instir(3 downto 0)="0010" then
                -- ST(-Z,Rr)
                s_stdecz <= '1';tpstack <= pin_addrrom + 1;
        s_addressram <= s_dupz - 1;s_addressram_dup <= s_dupz - 1;
                s_datainram <= gpr(conv_integer(pin_instir(7 downto 4)));
        s_datainram_dup <= gpr(conv_integer(pin_instir(7 downto 4)));
        elsif pin_instir(3 downto 0)="1001" then
                -- ST(Y+,Rr)
                s_styinc <= '1';tpstack <= pin_addrrom + 1;
                s_addressram <= s_dupy;s_addressram_dup <= s_dupy;
                s_datainram <= gpr(conv_integer(pin_instir(7 downto 4)));
        s_datainram_dup <= gpr(conv_integer(pin_instir(7 downto 4)));
        elsif pin_instir(3 downto 0)="1010" then
                -- ST(-Y,Rr)
                s_stdecy <= '1';tpstack <= pin_addrrom + 1;
        s_addressram <= s_dupy - 1;s_addressram_dup <= s_dupy - 1;
                s_datainram <= gpr(conv_integer(pin_instir(7 downto 4)));
        s_datainram_dup <= gpr(conv_integer(pin_instir(7 downto 4)));
        elsif pin_instir(3 downto 0)="1100" then
                -- ST(X,Rr)
                s_stx <= '1';tpstack <= pin_addrrom + 1;
                s_addressram <= s_dupx;s_addressram_dup <= s_dupx;
                s_datainram <= gpr(conv_integer(pin_instir(7 downto 4)));
        s_datainram_dup <= gpr(conv_integer(pin_instir(7 downto 4)));
        elsif pin_instir(3 downto 0)="1101" then
                -- ST(X+,Rr)
                s_stxinc <= '1';tpstack <= pin_addrrom + 1;
                s_addressram <= s_dupx;s_addressram_dup <= s_dupx;
                s_datainram <= gpr(conv_integer(pin_instir(7 downto 4)));
        s_datainram_dup <= gpr(conv_integer(pin_instir(7 downto 4)));
        elsif pin_instir(3 downto 0)="1110" then
                -- ST(-X,Rr)
                s_stdecx <= '1';  tpstack <= pin_addrrom + 1;
        s_addressram <= s_dupx - 1;s_addressram_dup <= s_dupx - 1;
                s_datainram <= gpr(conv_integer(pin_instir(7 downto 4)));
        s_datainram_dup <= gpr(conv_integer(pin_instir(7 downto 4)));
        elsif pin_instir(3 downto 0)="1111" then
                -- PUSH
                s_push <= '1';s_addresspc <= "000000000";
        end if;
elsif pin_instir(11 downto 9)="010" then
                if pin_instir(3 downto 0)="0000" then
                        -- COM
                        s_com <= '1';s_addresspc <= "000000000";
                        logicsel <= 3;logicsig <= '1';
                        opra <= gpr(conv_integer(pin_instir(7 downto 4)));
                elsif pin_instir(3 downto 0)="0001" then
```

```
            -- NEG
            s_neg <= '1';s_addresspc <= "000000000";
            arithsig <= '1';addsig <= '0';add_cflagin <= '1';
            oprb <= gpr(conv_integer(pin_instir(7 downto 4)));
            opra <= "00000000";
elsif pin_instir(3 downto 0)="0010" then
            -- SWAP
            s_swap <= '1';s_addresspc <= "000000000";
            opra <= gpr(conv_integer(pin_instir(7 downto 4)));
elsif pin_instir(3 downto 0)="0011" then
            -- INC
            s_inc <= '1';_addresspc <= "000000000";
            arithsig <= '1';addsig <= '1';add_cflagin <= '0';
            opra <= gpr(conv_integer(pin_instir(7 downto 4)));
            oprb <= "00000001";
elsif pin_instir(3 downto 0)="0101" then
            -- ASR
            s_asr <= '1';s_addresspc <= "000000000";
            shiftsel <= 2;shiftsig <= '1';
            opra <= gpr(conv_integer(pin_instir(7 downto 4)));
elsif pin_instir(3 downto 0)="0110" then
            -- LSR
            --s_lsr <= '1';s_addresspc <= "000000000";
            shiftsel <= 0;shiftsig <= '1';
            opra <= gpr(conv_integer(pin_instir(7 downto 4)));
elsif pin_instir(3 downto 0)="0111" then
            -- ROR
            s_ror <= '1';s_addresspc <= "000000000";
            shiftsel <= 1;shiftsig <= '1';
            opra <= gpr(conv_integer(pin_instir(7 downto 4)));
elsif pin_instir(3 downto 0)="1000" then
            if pin_instir(8 downto 7)="00" then
                    -- BSET,SEC,SEZ,SEN,SEV,SES,SEH,SET,SEI
                   s_bset <= '1';s_addresspc <= "000000000";
            elsif pin_instir(8 downto 7)="01" then
                    -- BCLR,CLC,CLZ,CLN,CLV,CLS,CLH,CLT,CLI
                   s_bclr <= '1';s_addresspc <= "000000000";
            elsif pin_instir(8 downto 7)="10" then
              -- RET,RETI
              if pin_instir(4)='0' then
                    -- RET
                    s_ret <= '1';s_addresspc <= stack0;
                    s_indirect <= '1';
              elsif pin_instir(4)='1' then
                    -- RETI
                    s_reti <= '1';s_addresspc <= stack0;
                    s_indirect <= '1';
              end if;
            elsif pin_instir(8 downto 7)="11" then
            -- SLEEP,BREAK,WDR,LPM,SPM
              if pin_instir(6 downto 4)="000" then
                    -- SLEEP
                   s_sleep <= '1';tpstack <= pin_addrrom + 1;
              elsif pin_instir(6 downto 4)="001" then
                    -- BREAK
                   s_break <= '1';tpstack <= pin_addrrom + 1;
              elsif pin_instir(6 downto 4)="010" then
                    -- WDR
                    s_wdr <= '1';
              elsif pin_instir(6 downto 4)="100" then
                    -- LPM
                    s_lpm <= '1';s_indirect <= '1';
                    s_addresspc <= s_zhigh(0) & s_dupz;
                    s_regd32inst <= "0000";
                    tpstack <= pin_addrrom + 1;
              elsif pin_instir(6 downto 4)="110" then
                    -- SPM
                    s_spm <= '1';tpstack <= pin_addrrom + 1;
                    pin_wrdatarom <= gpr(1) & gpr(0);
```

```
                    end if;
                end if;
            elsif pin_instir(3 downto 0)="1001" then
                if pin_instir(8 downto 4)="00000" then
                    -- IJMP
                    s_ijmp <= '1';s_indirect <= '1';
                    s_addresspc <= s_zhigh(0) & s_dupz;
                elsif pin_instir(8 downto 4)="10000" then
                    -- ICALL
                    s_icall <= '1';s_indirect <= '1';
                    s_addresspc <= s_zhigh(0) & s_dupz;
                    tpstack <= pin_addrrom + 1;
                end if;
            elsif pin_instir(3 downto 0)="1010" then
                -- DEC
                s_dec <= '1';s_addresspc <= "000000000";
                arithsig <= '1';addsig <= '0';add_cflagin <= '1';
                opra <= gpr(conv_integer(pin_instir(7 downto 4)));
                oprb <= "00000001";
            end if;
        elsif pin_instir(11 downto 9)="011" then
            if pin_instir(8)='0' then
                -- ADIW
                s_adiw <= '1';s_addresspc <= "000000000";
                AddLowHighCin <= '0';AddLowHighAddSub <= '1';
            oprab <= "00" & pin_instir(7 downto 6) & pin_instir(3 downto 0);
                if pin_instir(5 downto 4)="00" then
                    opra <= gpr(8);oprb <= gpr(9);
                elsif pin_instir(5 downto 4)="01" then
                    opra <= gpr(10);oprb <= gpr(11);
                elsif pin_instir(5 downto 4)="10" then
                    opra <= gpr(12);oprb <= gpr(13);
                elsif pin_instir(5 downto 4)="11" then
                    opra <= gpr(14);oprb <= gpr(15);
                end if;
            elsif pin_instir(8)='1' then
                -- SBIW
                s_sbiw <= '1';s_addresspc <= "000000000";
                AddLowHighCin <= '1';AddLowHighAddSub <= '0';
            oprab <= "00" & pin_instir(7 downto 6) & pin_instir(3 downto 0);
                if pin_instir(5 downto 4)="00" then
                    opra <= gpr(8);oprb <= gpr(9);
                elsif pin_instir(5 downto 4)="01" then
                    opra <= gpr(10);oprb <= gpr(11);
                elsif pin_instir(5 downto 4)="10" then
                    opra <= gpr(12);oprb <= gpr(13);
                elsif pin_instir(5 downto 4)="11" then
                    opra <= gpr(14);oprb <= gpr(15);
                end if;
            end if;
        elsif pin_instir(11 downto 9)="100" then
            if pin_instir(8)='0' then
                -- CBI
                s_sbicbi <= '1';s_addresspc <= "000000000";
                s_addrport <= '0' & pin_instir(7 downto 3);
            elsif pin_instir(8)='1' then
                -- SBIC
                s_sbics <= '1';is_sbic <= '1';is_sbis <= '0';
                bit_b <= pin_instir(2 downto 0);
                s_addrport <= '0' & pin_instir(7 downto 3);
                tpstack <= pin_addrrom + 1;
            end if;
        elsif pin_instir(11 downto 9)="101" then
            if pin_instir(8)='0' then
                -- SBI
                s_sbicbi <= '1';s_addresspc <= "000000000";
                s_addrport <= '0' & pin_instir(7 downto 3);
            elsif pin_instir(8)='1' then
                -- SBIS
```

```
                                s_sbics <= '1';is_sbis <= '1';is_sbic <= '0';
                                bit_b <= pin_instir(2 downto 0);
                                s_addrport <= '0' & pin_instir(7 downto 3);
                                tpstack <= pin_addrrom + 1;
                        end if;
                else
                        -- MUL
                        s_mul <= '1';s_addresspc <= "000000000";multiplysig <= '1';
                        s_sign <= '0';s_shift <= '0';
                        opra <= gpr(conv_integer(pin_instir(7 downto 4)));
                        oprb <= gpr(conv_integer(pin_instir(3 downto 0)));
                end if;
        when "1010" =>
                s_addresspc <= "000000000";
                if pin_instir(9)='0' then
                        if pin_instir(3)='0' then
                                -- LDD(Rd,Z+q)
                                s_lddzq <= '1';s_addresspc <= "000000000";
s_addressram <= s_dupz + ("00" & pin_instir(13) & pin_instir(11 downto 10) & pin_instir(2
downto 0));
                        elsif pin_instir(3)='1' then
                                -- LDD(Rd,Y+q)
                                s_lddyq <= '1';s_addresspc <= "000000000";
s_addressram <= s_dupy + ("00" & pin_instir(13) & pin_instir(11 downto 10) & pin_instir(2
downto 0));
                        end if;
                elsif pin_instir(9)='1' then
                        if pin_instir(3)='0' then
                                -- STD(Z+q,Rr)
                                s_stdzq <= '1';tpstack <= pin_addrrom + 1;
s_addressram <= s_dupy + ("00" & pin_instir(13) & pin_instir(11 downto 10) & pin_instir(2
downto 0));
s_addressram_dup <= s_dupy + ("00" & pin_instir(13) & pin_instir(11 downto 10) & pin_instir(2
downto 0));
                                s_datainram <= gpr(conv_integer(pin_instir(7 downto 4)));
                        s_datainram_dup <= gpr(conv_integer(pin_instir(7 downto 4)));
                        elsif pin_instir(3)='1' then
                                -- STD(Y+q,Rr)
                                s_stdyq <= '1';tpstack <= pin_addrrom + 1;
s_addressram <= s_dupy + ("00" & pin_instir(13) & pin_instir(11 downto 10) & pin_instir(2
downto 0));
s_addressram_dup <= s_dupy + ("00" & pin_instir(13) & pin_instir(11 downto 10) & pin_instir(2
downto 0));
                                s_datainram <= gpr(conv_integer(pin_instir(7 downto 4)));
                        s_datainram_dup <= gpr(conv_integer(pin_instir(7 downto 4)));
                        end if;
                end if;
        when "1011" =>
                s_addresspc <= "000000000";
                if pin_instir(11)='0' then
                        -- IN
                s_in <= '1';s_addrport <= pin_instir(10 downto 9) & pin_instir(3 downto 0);
                elsif pin_instir(11)='1' then
                        -- OUT
                s_out <= '1';s_addrport <= pin_instir(10 downto 9) & pin_instir(3 downto 0);
                end if;
        when "1100" =>
                -- RJMP
                s_rjmp <= '1';s_indirect <= '1';
                s_addresspc <= pin_addrrom + pin_instir(8 downto 0) + 1;
        when "1101" =>
                -- RCALL
                s_rcall <= '1';s_indirect <= '1';
                s_addresspc <= pin_addrrom + pin_instir(8 downto 0) + 1;
                tpstack <= pin_addrrom + 1;
        when "1110" =>
                s_addresspc <= "000000000";
                -- SER,LDI
                s_serldi <= '1';
```

```
                opra <= gpr(conv_integer(pin_instir(7 downto 4)));
                oprb <= pin_instir(11 downto 8) & pin_instir(3 downto 0);
        when "1111" =>
                if pin_instir(11 downto 10)="00" then
                        -- BRCS,BRLO,BREQ,BRMI,BRVS,BRLT,BRHS,BRTS,BRIE,BRBS
                        s_brcsbc <= '1';
                        -- BRCS - BRBS : Flag set?
                        if sr(conv_integer(pin_instir(2 downto 0)))='1' then  -- -> cabang
                                s_addresspc <= "00" & pin_instir(9 downto 3);
                        else                                                  -- -> tidak
                                s_addresspc <= "000000000";
                        end if;
                elsif pin_instir(11 downto 10)="01" then
                        -- BRCC,BRSH,BRNE,BRPL,BRVC,BRGE,BRHC,BRTC,BRID,BRBC
                        s_brcsbc <= '1';
                        -- BRCC - BRBC : Flag Cleared?
                        if sr(conv_integer(pin_instir(2 downto 0)))='0' then  -- -> cabang
                                s_addresspc <= "00" & pin_instir(9 downto 3);
                        else                                                  -- -> tidak
                                s_addresspc <= "000000000";
                        end if;
                elsif pin_instir(11 downto 10)="10" then
                        s_addresspc <= "000000000";
                        if pin_instir(9)='0' then     -- BLD  ; T->Rd
                          -- BLD,
                          s_bld <= '1';
                          opra <= gpr(conv_integer(pin_instir(7 downto 4)));
                          for i in 0 to 7 loop
                            if i=conv_integer(pin_instir(2 downto 0)) then
                                s_bldout(i) <= sr(6);
                            else
                                s_bldout(i) <= opra(i);
                            end if;
                 end loop;
                        elsif pin_instir(9)='1' then    -- BST   ; Rd->T
                          -- BST
                          s_bst <= '1';opra <= gpr(conv_integer(pin_instir(7 downto 4)));
                          for i in 0 to 7 loop
                            if i=conv_integer(pin_instir(2 downto 0)) then
                                s_bstout <= opra(i);
                            end if;
                 end loop;
                        end if;
                elsif pin_instir(11 downto 10)="11" then
                        s_addresspc <= "000000000";
                        if pin_instir(9)='0' then
                                -- SBRC
                                is_sbrc <= '1';is_sbrs <= '0';
                        elsif pin_instir(9)='1' then
                                -- SBRS
                                is_sbrc <= '0';is_sbrs <= '1';
                        else
                        end if;
                        s_sbrcs <= '1';
                        rr <= gpr(conv_integer(pin_instir(7 downto 4)));
                        bit_b <= pin_instir(2 downto 0);tpstack <= pin_addrrom + 1;
                end if;
        when others =>
        end case;
elsif (s_stat32='1' and s_statst='0' and s_statst2='0' and s_statst3='0' and s_statget='0' and
s_statstore='0' and s_statlpm='0' and s_statcpse='0' and s_statsbrcs='0' and s_statsbics='0'
and s_statspm0='0' and s_statspm='0' and s_statspm1='0' and s_statspm2='0') then    -- LDS
        s_addressram <= pin_instir(7 downto 0);
        s_datainram <= "00000000";s_addresspc <= "000000000";
elsif (s_stat32='0' and s_statst='1' and s_statst2='0' and s_statst3='0' and s_statget='0' and
s_statstore='0' and s_statlpm='0' and s_statcpse='0' and s_statsbrcs='0' and s_statsbics='0'
and s_statspm0='0' and s_statspm='0' and s_statspm1='0' and s_statspm2='0') then    -- ST
        s_addressram <= s_addressram_dup;s_datainram <= s_datainram_dup;
elsif (s_stat32='0' and s_statst='0' and s_statst2='1' and s_statst3='0' and s_statget='0' and
```

```
s_statstore='0' and s_statlpm='0' and s_statcpse='0' and s_statsbrcs='0' and s_statsbics='0'
and s_statspm0='0' and s_statspm='0' and s_statspm1='0' and s_statspm2='0') then    -- ST
        s_addressram <= s_addressram_dup;s_datainram <= s_datainram_dup;
elsif (s_stat32='0' and s_statst='0' and s_statst2='0' and s_statst3='1' and s_statget='0' and
s_statstore='0' and s_statlpm='0' and s_statcpse='0' and s_statsbrcs='0' and s_statsbics='0'
and s_statspm0='0' and s_statspm='0' and s_statspm1='0' and s_statspm2='0') then    -- ST
        s_addressram <= s_addressram_dup;s_datainram <= s_datainram_dup;
        s_indirect <='1';s_addresspc <= tpstack;
elsif (s_stat32='0' and s_statst='0' and s_statst2='0' and s_statst3='0' and s_statget='1' and
s_statstore='0' and s_statlpm='0' and s_statcpse='0' and s_statsbrcs='0' and s_statsbics='0'
and s_statspm0='0' and s_statspm='0' and s_statspm1='0' and s_statspm2='0') then
        -- detect STS
s_addressram <= pin_instir(7 downto 0);s_datainram <= gpr(conv_integer(s_regd32inst));
s_addresspc <= "000000000";
elsif (s_stat32='0' and s_statst='0' and s_statst2='0' and s_statst3='0' and s_statget='0' and
s_statstore='1' and s_statlpm='0' and s_statcpse='0' and s_statsbrcs='0' and s_statsbics='0'
and s_statspm0='0' and s_statspm='0' and s_statspm1='0' and s_statspm2='0') then
s_addressram <= pin_instir(7 downto 0);s_datainram <= gpr(conv_integer(s_regd32inst));
s_addresspc <= "000000000";
elsif (s_stat32='0' and s_statst='0' and s_statst2='0' and s_statst3='0' and s_statget='0' and
s_statstore='0' and s_statlpm='1' and s_statcpse='0' and s_statsbrcs='0' and s_statsbics='0'
and s_statspm0='0' and s_statspm='0' and s_statspm1='0' and s_statspm2='0') then
        s_indirect <= '1';s_addresspc <= tpstack;
        if s_dupz(0)='0' then
          s_instir <= pin_instir(7 downto 0);
        else
          s_instir <= pin_instir(15 downto 8);
        end if;
        s_zinclpm <= '1';
elsif (s_stat32='0' and s_statst='0' and s_statst2='0' and s_statst3='0' and s_statget='0' and
s_statstore='0' and s_statlpm='0' and s_statcpse='1' and s_statsbrcs='0' and s_statsbics='0'
and s_statspm0='0' and s_statspm='0' and s_statspm1='0' and s_statspm2='0') then
        if rd=rr then
          s_indirect <= '1';s_addresspc <= tpstack + 1;
        else
          s_indirect <= '1';s_addresspc <= tpstack;
        end if;
elsif (s_stat32='0' and s_statst='0' and s_statst2='0' and s_statst3='0' and s_statget='0' and
s_statstore='0' and s_statlpm='0' and s_statcpse='0' and s_statsbrcs='1' and s_statsbics='0'
and s_statspm0='0' and s_statspm='0' and s_statspm1='0' and s_statspm2='0') then
        if (is_sbrc='1' and is_sbrs='0') then                -- SBRC ; clear?
                if rr(conv_integer(bit_b))='0' then  -- Skip
                        s_indirect <= '1';s_addresspc <= tpstack + 1;
                else
                        s_indirect <= '1';s_addresspc <= tpstack;
                end if;
        elsif (is_sbrc='0' and is_sbrs='1') then        -- SBRS ; set?
                if rr(conv_integer(bit_b))='1' then  -- Skip
                        s_indirect <= '1';s_addresspc <= tpstack + 1;
                else
                        s_indirect <= '1';s_addresspc <= tpstack;
                end if;
        else
        end if;
elsif (s_stat32='0' and s_statst='0' and s_statst2='0' and s_statst3='0' and s_statget='0' and
s_statstore='0' and s_statlpm='0' and s_statcpse='0' and s_statsbrcs='0' and s_statsbics='1'
and s_statspm0='0' and s_statspm='0' and s_statspm1='0' and s_statspm2='0') then
        if (is_sbic='1' and is_sbis='0') then            -- SBIC
                        if (s_addrport="011001") then                -- Ain $19
                                if pin_Ainport(conv_integer(bit_b))='0' then
                                        s_indirect <= '1';s_addresspc <= tpstack + 1;
                                else
                                        s_indirect <= '1';s_addresspc <= tpstack;
                                end if;
                        elsif (s_addrport="010110") then             -- Bin $16
                                if pin_Binport(conv_integer(bit_b))='0' then
                                        s_indirect <= '1';s_addresspc <= tpstack + 1;
                                else
                                        s_indirect <= '1';s_addresspc <= tpstack;
```

```
                        end if;
                    end if;
        elsif (is_sbic='0' and is_sbis='1') then        -- SBIS
                    if (pin_instir(2 downto 0)="011001") then   -- Ain $19
                        if pin_Ainport(conv_integer(bit_b))='1' then
                            s_indirect <= '1';s_addresspc <= tpstack + 1;
                        else
                            s_indirect <= '1';s_addresspc <= tpstack;
                        end if;
                    elsif (pin_instir(2 downto 0)="010110") then          -- Bin $16
                        if pin_Binport(conv_integer(bit_b))='1' then
                            s_indirect <= '1';s_addresspc <= tpstack + 1;
                        else
                            s_indirect <= '1';s_addresspc <= tpstack;
                        end if;
                    end if;
        else
        end if;
elsif (s_stat32='0' and s_statst='0' and s_statst2='0' and s_statst3='0' and s_statget='0' and
s_statstore='0' and s_statlpm='0' and s_statcpse='0' and s_statsbrcs='0' and s_statsbics='0' and
s_statspm0='1' and s_statspm='0' and s_statspm1='0' and s_statspm2='0') then
s_indirect <= '1';s_addresspc <= s_zhigh(0) & s_dupz;pin_wrdatarom <= gpr(1) & gpr(0);
elsif (s_stat32='0' and s_statst='0' and s_statst2='0' and s_statst3='0' and s_statget='0' and
s_statstore='0' and s_statlpm='0' and s_statcpse='0' and s_statsbrcs='0' and s_statsbics='0'
and s_statspm0='0' and s_statspm='1' and s_statspm1='0' and s_statspm2='0') then
s_indirect <= '1';s_addresspc <= s_zhigh(0) & s_dupz;pin_wrdatarom <= gpr(1) & gpr(0);
elsif (s_stat32='0' and s_statst='0' and s_statst2='0' and s_statst3='0' and s_statget='0' and
s_statstore='0' and s_statlpm='0' and s_statcpse='0' and s_statsbrcs='0' and s_statsbics='0'
and s_statspm0='0' and s_statspm='0' and s_statspm1='1' and s_statspm2='0') then
s_indirect <= '1'; s_addresspc <= s_zhigh(0) & s_dupz;pin_wrdatarom <= gpr(1) & gpr(0);
elsif (s_stat32='0' and s_statst='0' and s_statst2='0' and s_statst3='0' and s_statget='0' and
s_statstore='0' and s_statlpm='0' and s_statcpse='0' and s_statsbrcs='0' and s_statsbics='0' and
s_statspm0='0' and s_statspm='0' and s_statspm1='0' and s_statspm2='1') then
s_indirect <= '1';s_addresspc <= tpstack;
else
-- (sleep,break)
end if;
end process;
pcounter: PCUnit
port map (pin_clk,pin_rst,s_enablepc,pin_addrrom,s_addresspc,s_indirect,s_int0int,s_timerint);
s_arithout <= add_out;
addersubtracter: lpm_add_sub
                generic map (lpm_width=>8)
                port map (dataa=>opra,datab=>oprb,cin=>add_cflagin,add_sub=>addsig,
                result=>add_out,cout=>add_cout,overflow=>add_overflow);
AddLowPart: AddLowUnit port map
(opra,oprab,AddLowHighCin,AddLowHighAddSub,s_addsubiwout1,s_AddLowCout,s_AddLowOv);

AddHighPart: AddHighUnit port map
(oprb,oprab,AddLowHighCin,AddLowHighAddSub,s_addsubiwout2,s_AddHighCout,s_AddHighOv);

shifter_enable <= shiftsig;shifter_in <= opra;shifter_selop <= shiftsel;shifter_cflagin <= sr(0);
--shifter_cflagout
s_shiftout <= shifter_out;
ShiftPart: ShifterUnit port map
(shifter_enable,shifter_in,shifter_selop,shifter_cflagin,shifter_cflagout,shifter_out);

logic_enable <= logicsig;logic_opra <= opra;logic_oprb <= oprb;logic_selop <= logicsel;
s_logicout <= logic_out;
LogicPart: LogicUnit
        port map (logic_enable,logic_opra,logic_oprb,logic_selop,logic_out);

swap_in <= opra;
SwapPart: SwapUnit
        port map (swap_in,swap_out);
s_swapout <= swap_out;mult_sign <= s_sign;mult_a <= opra;mult_b <= oprb;
mult_shift <= s_shift;s_multouthigh <= mult_resulthigh;s_multoutlow <= mult_resultlow;
s_multcarry<=mult_carry;s_multsuouthigh<=multsu_resulthigh;
s_multsuoutlow<=multsu_resultlow;s_multsucarry <= multsu_carry;
```

```
MultPart: MultUnit
port map(mult_a,mult_b,mult_sign,mult_shift,mult_resulthigh,mult_resultlow,mult_carry);
MultsuPart: MulsuUnit
port map(mult_a,mult_b,mult_shift,multsu_resulthigh,multsu_resultlow,multsu_carry);
RAMUnitPart: RAMUnit port map (s_addressram,s_enastclk,s_datainram,s_outram);
s_dupz <= gpr(14);s_zhigh <= gpr(15);s_dupx <= gpr(10);s_dupy <= gpr(12);
buffExtIntPart: buffExtInt port map (ena_extirq,Ainport7,s_int0int);
Ainport7 <= pin_Ainport(7);

process(pin_rst,pin_clk)
begin
if pin_rst='0' then
        state <= exestate;
        s_statsbics <= '0';s_statsbrcs <= '0';s_statcpse <= '0';s_stat32 <= '0';s_statst <= '0';
        s_statst2 <= '0';s_statst3 <= '0';s_statget <= '0';s_statstore <= '0';s_statlpm <= '0';
        s_statspm0 <= '0';s_statspm <= '0';s_statspm1 <= '0';s_statspm2 <= '0';
        pin_Coutport <= "00000000";pin_Doutport <= "00000000";
        s_enablepc <= '1';sr <= "00000000";
        stack7 <= "000000000";stack6 <= "000000000";stack5 <= "000000000";
        stack4 <= "000000000";stack3 <= "000000000";stack2 <= "000000000";
        stack1 <= "000000000";stack0 <= "000000000";
        pin_wdrrestart <= '1';
elsif pin_clk'event and pin_clk='0' then
 case state is
  when exestate =>
        s_statsbics <= '0';s_statsbrcs <= '0';s_statcpse <= '0';s_stat32 <= '0';s_statst <= '0';
        s_statst2 <= '0';s_statst3 <= '0';s_statget <= '0';s_statstore <= '0';s_statlpm <= '0';
        s_statspm0 <= '0';s_statspm <= '0';s_statspm1 <= '0';s_statspm2 <= '0';
        pin_wdrrestart <= '1';s_enablepc <= '1';pin_wrtcnt <= '0';
        if (s_sleep='1' or s_break='1') then
                state <= standbystate;
                stack7 <= stack6;stack6 <= stack5;stack5 <= stack4;stack4 <= stack3;
                stack3 <= stack2;stack2 <= stack1;stack1 <= stack0;stack0 <= tpstack;
                s_statsbics <= '1';s_statsbrcs <= '1';s_statcpse <= '1';s_stat32 <= '1';
                s_statst <= '1';s_statst2 <= '1';s_statst3 <= '1';s_statget <= '1';
                s_statstore <= '1';s_statlpm <= '1';s_statspm0 <= '1';s_statspm <= '1';
                s_statspm1 <= '1';s_statspm2 <= '1';state <= standbystate;
        elsif logicsig='1' then                 --(and,andi,or,ori,eor,com)
                gpr(conv_integer(pin_instir(7 downto 4))) <= s_logicout;
                -- AND(TST),ANDI(CBR),OR,ORI(SBR),EOR(CLR)
                -- COM
        if (s_andtst='1' or s_andicbr='1' or s_or='1' or s_orisbr='1' or s_eorclr='1') then
                -- SV=0,NZ
                  sr(4) <= sr(2) xor sr(3);
                  sr(3) <= '0';
                  sr(2) <= s_logicout(7);
sr(1) <= not s_logicout(7) and not s_logicout(6) and not s_logicout(5) and not s_logicout(4) and
not s_logicout(3) and not s_logicout(2) and not s_logicout(1) and not s_logicout(0);
                elsif (s_com='1') then
                -- SV=0,NZC=1
                  sr(4) <= sr(2) xor sr(3);
                  sr(3) <= '0';
                  sr(2) <= s_logicout(7);
sr(1) <= not s_logicout(7) and not s_logicout(6) and not s_logicout(5) and not s_logicout(4) and
        not s_logicout(3) and not s_logicout(2) and not s_logicout(1) and not s_logicout(0);
                  sr(0) <= '1';
                end if;
        elsif shiftsig='1' then                 --(lsr,ror,asr)
                gpr(conv_integer(pin_instir(7 downto 4))) <= s_shiftout;
                -- ASR,LSR,ROR
                if (s_asr='1' or s_lsr='1' or s_ror='1') then
                -- SVNZC
                  sr(4) <= sr(3) xor sr(2);
                  sr(3) <= sr(2) xor sr(0);
                  sr(2) <= s_shiftout(7);
sr(1) <= not s_shiftout(7) and not s_shiftout(6) and not s_shiftout(5) and not s_shiftout(4) and
not s_shiftout(3) and not s_shiftout(2) and not s_shiftout(1) and not s_shiftout(0);
                  sr(0) <= opra(0);
```

```
                end if;
        elsif arithsig='1' then                 --(adc,rol,add,lsl,inc,sbc,sbci,sub,subi,dec,neg)
                gpr(conv_integer(pin_instir(7 downto 4))) <= s_arithout;
                -- ADD(LSL),ADC(ROL),INC,SBC,SBCI,SUB,SUBI,DEC,NEG
                if (s_addlsl='1' or s_adcrol='1') then
                -- HSVNZC
sr(5)<=(opra(3) and oprb(3)) or (oprb(3) and not s_arithout(3)) or (not s_arithout(3) and
opra(3));
          sr(4) <= sr(3) xor sr(2);
sr(3) <= (opra(7) and oprb(7) and not s_arithout(7)) or (not opra(7) and not oprb(7) and
s_arithout(7));
                  sr(2) <= s_arithout(7);
sr(1) <= not s_arithout(7) and not s_arithout(6) and not s_arithout(5) and not s_arithout(4) and
         not s_arithout(3) and not s_arithout(2) and not s_arithout(1) and not s_arithout(0);
                  sr(0) <= (opra(7) and oprb(7)) or (oprb(7) and not s_arithout(7)) or (not
s_arithout(7) and opra(7));
                elsif (s_inc='1') then
                -- SVNZ
                  sr(4) <= sr(3) xor sr(2);
sr(3) <= s_arithout(7) and not s_arithout(6) and not s_arithout(5) and not s_arithout(4) and
         not s_arithout(3) and not s_arithout(2) and not s_arithout(1) and not s_arithout(0);
                  sr(2) <= s_arithout(7);
sr(1) <= not s_arithout(7) and not s_arithout(6) and not s_arithout(5) and not s_arithout(4) and
         not s_arithout(3) and not s_arithout(2) and not s_arithout(1) and not s_arithout(0);
                elsif (s_sbc='1' or s_sbci='1') then
                -- HSVNZC
sr(5) <= (not opra(3) and oprb(3)) or (oprb(3) and s_arithout(3)) or (s_arithout(3) and not
opra(3));
                  sr(4) <= sr(3) xor sr(2);
sr(3) <= (opra(7) and not oprb(7) and not s_arithout(7)) or (not opra(7) and oprb(7) and
s_arithout(7));
                  sr(2) <= s_arithout(7);
sr(1) <= not s_arithout(7) and not s_arithout(6) and not s_arithout(5) and not s_arithout(4) and
  not s_arithout(3) and not s_arithout(2) and not s_arithout(1) and not s_arithout(0) and sr(1);
sr(0) <= (not opra(7) and oprb(7)) or (oprb(7) and s_arithout(7)) or (s_arithout(7) and not
opra(7));
                elsif (s_sub='1' or s_subi='1') then
                -- HSVNZC
sr(5) <= (not opra(3) and oprb(3)) or (oprb(3) and s_arithout(3)) or (s_arithout(3) and not
opra(3));
                  sr(4) <= sr(3) xor sr(2);
sr(3) <= (opra(7) and not oprb(7) and not s_arithout(7)) or (not opra(7) and oprb(7) and
s_arithout(7));
                  sr(2) <= s_arithout(7);
sr(1) <= not s_arithout(7) and not s_arithout(6) and not s_arithout(5) and not s_arithout(4) and
           not s_arithout(3) and not s_arithout(2) and not s_arithout(1) and not s_arithout(0);
sr(0) <= (not opra(7) and oprb(7)) or (oprb(7) and s_arithout(7)) or (s_arithout(7) and not
opra(7));
                elsif (s_dec='1') then
                -- SVNZ
                  sr(4) <= sr(3) xor sr(2);
sr(3) <= not s_arithout(7) and s_arithout(6) and s_arithout(5) and s_arithout(4) and
           s_arithout(3) and s_arithout(2) and s_arithout(1) and s_arithout(0);
                  sr(2) <= s_arithout(7);
sr(1) <= not s_arithout(7) and not s_arithout(6) and not s_arithout(5) and not s_arithout(4) and
         not s_arithout(3) and not s_arithout(2) and not s_arithout(1) and not s_arithout(0);
                elsif (s_neg='1') then
                -- HSVNZC
                  sr(5) <= s_arithout(3) or opra(3);
                  sr(4) <= sr(3) xor sr(2);
sr(3) <= s_arithout(7) and not s_arithout(6) and not s_arithout(5) and not s_arithout(4) and
           not s_arithout(3) and not s_arithout(2) and not s_arithout(1) and not s_arithout(0);
                  sr(2) <= s_arithout(7);
sr(1) <= not s_arithout(7) and not s_arithout(6) and not s_arithout(5) and not s_arithout(4) and
           not s_arithout(3) and not s_arithout(2) and not s_arithout(1) and not s_arithout(0);
sr(0) <= s_arithout(7) and s_arithout(6) and s_arithout(5) and s_arithout(4) and
           s_arithout(3) and s_arithout(2) and s_arithout(1) and s_arithout(0);
                end if;
        elsif (s_adiw='1' or s_sbiw='1') then
```

```
                    if pin_instir(5 downto 4)="00" then
                            gpr(8) <= s_addsubiwout1;gpr(9) <= s_addsubiwout2;
                    elsif pin_instir(5 downto 4)="01" then
                            gpr(10) <= s_addsubiwout1;gpr(11) <= s_addsubiwout2;
                    elsif pin_instir(5 downto 4)="10" then
                            gpr(12) <= s_addsubiwout1;gpr(13) <= s_addsubiwout2;
                    elsif pin_instir(5 downto 4)="11" then
                            gpr(14) <= s_addsubiwout1;gpr(15) <= s_addsubiwout2;
                    end if;
                    -- ADIW,SBIW
                    -- HSVNZC
                      sr(5) <= s_arithout(3) or opra(3);
                      sr(4) <= sr(3) xor sr(2);
sr(3) <= s_arithout(7) and not s_arithout(6) and not s_arithout(5) and not s_arithout(4) and
not s_arithout(3) and not s_arithout(2) and not s_arithout(1) and not s_arithout(0);
                      sr(2) <= s_arithout(7);
sr(1) <= not s_arithout(7) and not s_arithout(6) and not s_arithout(5) and not s_arithout(4) and
not s_arithout(3) and not s_arithout(2) and not s_arithout(1) and not s_arithout(0);
sr(0) <= s_arithout(7) and s_arithout(6) and s_arithout(5) and s_arithout(4) and
            s_arithout(3) and s_arithout(2) and s_arithout(1) and s_arithout(0);
          elsif s_swap='1' then
                    gpr(conv_integer(pin_instir(7 downto 4))) <= s_swapout;
          elsif s_mov='1' then
          gpr(conv_integer(pin_instir(7 downto 4))) <= gpr(conv_integer(pin_instir(3 downto 0)));
          elsif s_movw='1' then
                    gpr(0) <= opra;gpr(1) <= oprb;
          elsif multiplysig='1' then
                    if (s_mul='1' or s_fmul='1' or s_muls='1' or s_fmuls='1') then
                            gpr(1) <= s_multouthigh;gpr(0) <= s_multoutlow;
                    else
                            gpr(1) <= s_multsuouthigh;gpr(0) <= s_multsuoutlow;
                    end if;
                    -- MUL,MULS,MULSU
                    if (s_mul='1' or s_muls='1' or s_mulsu='1') then
                    -- ZC
sr(1) <= not s_multouthigh(7) and not s_multouthigh(6) and not s_multouthigh(5) and not
s_multouthigh(4) and not s_multouthigh(3) and not s_multouthigh(2) and
not s_multouthigh(1) and not s_multouthigh(0) and not s_multoutlow(7) and
not s_multoutlow(6) and not s_multoutlow(5) and not s_multoutlow(4) and
not s_multoutlow(3) and not s_multoutlow(2) and not s_multoutlow(1) and not s_multoutlow(0);
                      sr(0) <= s_multouthigh(7);
                    elsif (s_fmul='1' or s_fmuls='1' or s_fmulsu='1') then
                    -- FMUL,FMULS,FMULSU
                    -- ZC
sr(1) <= not s_multouthigh(7) and not s_multouthigh(6) and not s_multouthigh(5) and not
s_multouthigh(4) and not s_multouthigh(3) and not s_multouthigh(2) and
not s_multouthigh(1) and not s_multouthigh(0) and not s_multoutlow(7) and
not s_multoutlow(6) and not s_multoutlow(5) and not s_multoutlow(4) and
not s_multoutlow(3) and not s_multoutlow(2) and not s_multoutlow(1) and not s_multoutlow(0);
                      sr(0) <= s_multcarry;
                    end if;
          elsif s_bset='1' then
                    sr(conv_integer(pin_instir(6 downto 4))) <= '1';
          elsif s_bclr='1' then
                    sr(conv_integer(pin_instir(6 downto 4))) <= '0';
          elsif s_bld='1' then                              -- T=>Rd(b)
                    gpr(conv_integer(pin_instir(7 downto 4))) <= s_bldout;
          elsif s_bst='1' then                              -- Rd(b)=>T
                    -- BST
                    sr(6) <= s_bstout;
          elsif s_serldi='1' then
                    gpr(conv_integer(pin_instir(7 downto 4))) <= oprb;
          elsif (s_ldx='1' or s_ldy='1' or s_ldz='1' or s_lddyq='1' or s_lddzq='1') then
                    gpr(conv_integer(pin_instir(7 downto 4))) <= s_outram;
          elsif (s_lds='1') then
                    s_statsbics <= '0';s_statsbrcs <= '0';s_statcpse <= '0';s_stat32 <= '1';
                    s_statst <= '0';s_statst2 <= '0';s_statst3 <= '0';s_statget <= '0';
                    s_statstore <= '0';s_statlpm <= '0';s_statspm0 <= '0';s_statspm <= '0';
                    s_statspm1 <= '0';s_statspm2 <= '0';state <= exestate32;
```

```
elsif (s_lpm='1' or s_lpmz='1') then
        s_statsbics <= '0';s_statsbrcs <= '0';s_statcpse <= '0';s_stat32 <= '0';
        s_statst <= '0';s_statst2 <= '0';s_statst3 <= '0';s_statget <= '0';
        s_statstore <= '0';s_statlpm <= '1';s_statspm0 <= '0';s_statspm <= '0';
        s_statspm1 <= '0';s_statspm2 <= '0';state <= lpmstate;
elsif (s_lpmzinc='1') then
        s_statsbics <= '0';s_statsbrcs <= '0';s_statcpse <= '0';s_stat32 <= '0';
        s_statst <= '0';s_statst2 <= '0';s_statst3 <= '0';s_statget <= '0';
        s_statstore <= '0';s_statlpm <= '1';s_statspm0 <= '0';s_statspm <= '0';
        s_statspm1 <= '0';s_statspm2 <= '0';state <= lpmstate;
elsif (s_ldxinc='1') then
        gpr(conv_integer(pin_instir(7 downto 4))) <= s_outram;
        gpr(10) <= s_dupx + 1;
elsif (s_ldyinc='1') then
        gpr(conv_integer(pin_instir(7 downto 4))) <= s_outram;
        gpr(12) <= s_dupy + 1;
elsif (s_ldzinc='1') then
        gpr(conv_integer(pin_instir(7 downto 4))) <= s_outram;
        gpr(14) <= s_dupz + 1;
elsif (s_lddecx='1') then
        gpr(conv_integer(pin_instir(7 downto 4))) <= s_outram;
        gpr(10) <= s_dupx - 1;
elsif (s_lddecy='1') then
        gpr(conv_integer(pin_instir(7 downto 4))) <= s_outram;
        gpr(12) <= s_dupy - 1;
elsif (s_lddecz='1') then
        gpr(conv_integer(pin_instir(7 downto 4))) <= s_outram;
        gpr(14) <= s_dupz - 1;
elsif (s_stx='1' or s_sty='1' or s_stz='1' or s_stdyq='1' or s_stdzq='1') then
        s_statsbics <= '0';s_statsbrcs <= '0';s_statcpse <= '0';s_stat32 <= '0';
        s_statst <= '1';s_statst2 <= '0';s_statst3 <= '0';s_statget <= '0';
        s_statstore <= '0';s_statlpm <= '0';s_statspm0 <= '0';s_statspm <= '0';
        s_statspm1 <= '0';s_statspm2 <= '0';state <= store2;s_enastclk <= '1';
elsif (s_sts='1') then
        s_statsbics <= '0';s_statsbrcs <= '0';s_statcpse <= '0';s_stat32 <= '0';
        s_statst <= '0';s_statst2 <= '0';s_statst3 <= '0';s_statget <= '1';
        s_statstore <= '0';s_statlpm <= '0';s_statspm0 <= '0';s_statspm <= '0';
        s_statspm1 <= '0';s_statspm2 <= '0';state <= getaddress;
elsif (s_stxinc='1') then
        s_statsbics <= '0';s_statsbrcs <= '0';s_statcpse <= '0';s_stat32 <= '0';
        s_statst <= '1';s_statst2 <= '0';s_statst3 <= '0';s_statget <= '0';
        s_statstore <= '0';s_statlpm <= '0';s_statspm0 <= '0';s_statspm <= '0';
        s_statspm1 <= '0';s_statspm2 <= '0';state <= store2;
        gpr(10) <= s_dupx + 1;s_enastclk <= '1';
elsif (s_styinc='1') then
        s_statsbics <= '0';s_statsbrcs <= '0';s_statcpse <= '0';s_stat32 <= '0';
        s_statst <= '1';s_statst2 <= '0';s_statst3 <= '0';s_statget <= '0';
        s_statstore <= '0';s_statlpm <= '0';s_statspm0 <= '0';s_statspm <= '0';
        s_statspm1 <= '0';s_statspm2 <= '0';state <= store2;
        gpr(12) <= s_dupy + 1;s_enastclk <= '1';
elsif (s_stzinc='1') then
        s_statsbics <= '0';s_statsbrcs <= '0';s_statcpse <= '0';s_stat32 <= '0';
        s_statst <= '1';s_statst2 <= '0';s_statst3 <= '0';s_statget <= '0';
        s_statstore <= '0';s_statlpm <= '0';s_statspm0 <= '0';s_statspm <= '0';
        s_statspm1 <= '0';s_statspm2 <= '0';state <= store2;g
        pr(14) <= s_dupz + 1;s_enastclk <= '1';
elsif (s_stdecx='1') then
        s_statsbics <= '0';s_statsbrcs <= '0';s_statcpse <= '0';s_stat32 <= '0';
        s_statst <= '1';s_statst2 <= '0';s_statst3 <= '0';s_statget <= '0';
        s_statstore <= '0';s_statlpm <= '0';s_statspm0 <= '0';s_statspm <= '0';
        s_statspm1 <= '0';s_statspm2 <= '0';state <= store2;
        gpr(10) <= s_dupx - 1;s_enastclk <= '1';
elsif (s_stdecy='1') then
        s_statsbics <= '0';s_statsbrcs <= '0';s_statcpse <= '0';s_stat32 <= '0';
        s_statst <= '1';s_statst2 <= '0';s_statst3 <= '0';s_statget <= '0';
        s_statstore <= '0';s_statlpm <= '0';s_statspm0 <= '0';s_statspm <= '0';
        s_statspm1 <= '0';s_statspm2 <= '0';state <= store2;
        gpr(12) <= s_dupy - 1;s_enastclk <= '1';
elsif (s_stdecz='1') then
```

```
            s_statsbics <= '0';s_statsbrcs <= '0';s_statcpse <= '0';s_stat32 <= '0';
            s_statst <= '1';s_statst2 <= '0';s_statst3 <= '0';s_statget <= '0';
            s_statstore <= '0';s_statlpm <= '0';s_statspm0 <= '0';s_statspm <= '0';
            s_statspm1 <= '0';s_statspm2 <= '0';state <= store2;
            gpr(14) <= s_dupz - 1;s_enastclk <= '1';
    elsif (s_in='1') then                                       -- 64 I/O address
            if s_addrport="111011" then         -- 0x3B (GICR;INT0 enable;INT0;bit-6)
                    gpr(conv_integer(pin_instir(7 downto 4))) <= s_gicr;
            elsif s_addrport="111001" then      -- 0x39 (TIMSK;Timer0 enable;toie;bit-0)
                    gpr(conv_integer(pin_instir(7 downto 4))) <= s_timsk;
            elsif s_addrport="110011" then
            -- 0x33 (TCCR0;Timer0 mode;cs02 bit-2,cs01 bit-1,cs00 bit-0)
                    gpr(conv_integer(pin_instir(7 downto 4))) <= s_tccr;
            elsif s_addrport="100001" then      -- 0x21 (WDTCR;WDE bit-3,WDP bit-2 sd 0)
                    gpr(conv_integer(pin_instir(7 downto 4))) <= s_wdtcr;
            elsif s_addrport="111111" then                  -- 0x3F
                    gpr(conv_integer(pin_instir(7 downto 4))) <= sr;
            elsif s_addrport="011011" then                  -- port A in (pinA:$1B)
                    gpr(conv_integer(pin_instir(7 downto 4))) <= pin_Ainport;
            elsif s_addrport="011000" then                  -- port B in (pinB:$18)
                    gpr(conv_integer(pin_instir(7 downto 4))) <= pin_Binport;
            end if;
    elsif (s_out='1') then                                      -- 64 I/O address
            if s_addrport="111011" then                 -- 0x3B (GICR;INT0 enable;bit-6)
                    s_gicr <= gpr(conv_integer(pin_instir(7 downto 4)));
            elsif s_addrport="111001" then      -- 0x39 (TIMSK;Timer0 enable;toie;bit-0)
                    s_timsk <= gpr(conv_integer(pin_instir(7 downto 4)));
            elsif s_addrport="110011" then
            -- 0x33 (TCCR0;Timer0 mode;cs02 bit-2,cs01 bit-1,cs00 bit-0)
                    s_tccr <= gpr(conv_integer(pin_instir(7 downto 4)));
            elsif s_addrport="110010" then              -- 0x32 (TCNT0;Timer0 count)
                    pin_tcnt <= gpr(conv_integer(pin_instir(7 downto 4)));
                    pin_wrtcnt <= '1';
            elsif s_addrport="111111" then              -- 0x3f (SREG)
                    sr <= gpr(conv_integer(pin_instir(7 downto 4)));
            elsif s_addrport="100001" then      -- 0x21 (WDTCR;WDE bit-3,WDP bit-2 sd 0)
                    s_wdtcr <= gpr(conv_integer(pin_instir(7 downto 4)));
            else
            if s_addrport="010101" then                     -- port C out (portC:$15)
                        pin_Coutport <= gpr(conv_integer(pin_instir(7 downto 4)));
                    elsif s_addrport="010010" then          -- port D out (portD:$12)
                        pin_Doutport <= gpr(conv_integer(pin_instir(7 downto 4)));
            end if;
            end if;
    elsif (s_wdr='1') then
            pin_wdrrestart <= '0';
    elsif (s_sbicbi='1') then                               -- 32 lower I/O address
        if pin_instir(9)='0' then
          if s_addrport="010101" then                       -- port C out (portC:$15)
                    pin_Coutport(conv_integer(pin_instir(2 downto 0))) <= '0';
          elsif s_addrport="010010" then                    -- port D out (portD:$12)
                    pin_Doutport(conv_integer(pin_instir(2 downto 0))) <= '0';
          end if;
        else
          if s_addrport="010101" then                       -- port C out (portC:$15)
                    pin_Coutport(conv_integer(pin_instir(2 downto 0))) <= '1';
          elsif s_addrport="010010" then                    -- port D out (portD:$12)
                    pin_Doutport(conv_integer(pin_instir(2 downto 0))) <= '1';
          end if;
        end if;
    elsif (s_sbics='1') then
            s_statsbics <= '1';s_statsbrcs <= '0';s_statcpse <= '0';s_stat32 <= '0';
            s_statst <= '0';s_statst2 <= '0';s_statst3 <= '0';s_statget <= '0';
            s_statstore <= '0';s_statlpm <= '0';s_statspm0 <= '0';s_statspm <= '0';
            s_statspm1 <= '0';s_statspm2 <= '0';state <= sbicsstate;
    elsif (s_brcsbc='1') then
    elsif (s_cpse='1') then
            s_statsbics <= '0';s_statsbrcs <= '0';s_statcpse <= '1';s_stat32 <= '0';
            s_statst <= '0';s_statst2 <= '0';s_statst3 <= '0';s_statget <= '0';
```

```
                s_statstore <= '0';s_statlpm <= '0';s_statspm0 <= '0';s_statspm <= '0';
                s_statspm1 <= '0';s_statspm2 <= '0';state <= cpsestate;
        elsif (s_cp='1' or s_cpi='1') then
                -- HSVNZC
sr(5) <= (not opra(3) and oprb(3)) or (oprb(3) and s_arithout(3)) or (s_arithout(3) and not
opra(3));
                sr(4) <= sr(3) xor sr(2);
sr(3) <= (opra(7) and not oprb(7) and not s_arithout(7)) or (not opra(7) and oprb(7) and
        s_arithout(7));
                sr(2) <= s_arithout(7);
sr(1) <= not s_arithout(7) and not s_arithout(6) and not s_arithout(5) and not s_arithout(4) and
         not s_arithout(3) and not s_arithout(2) and not s_arithout(1) and not s_arithout(0);
sr(0) <= (not opra(7) and oprb(7)) or (oprb(7) and s_arithout(7)) or (s_arithout(7) and not
opra(7));
        elsif (s_cpc='1') then
                -- HSVNZC
sr(5) <= (not opra(3) and oprb(3)) or (oprb(3) and s_arithout(3)) or (s_arithout(3) and not
opra(3));
                sr(4) <= sr(3) xor sr(2);
sr(3) <= (opra(7) and not oprb(7) and not s_arithout(7)) or (not opra(7) and oprb(7) and
        s_arithout(7));
                sr(2) <= s_arithout(7);
sr(1) <= not s_arithout(7) and not s_arithout(6) and not s_arithout(5) and not s_arithout(4) and
         not s_arithout(3) and not s_arithout(2) and not s_arithout(1) and not s_arithout(0) and
sr(1);
sr(0) <= (not opra(7) and oprb(7)) or (oprb(7) and s_arithout(7)) or (s_arithout(7) and not
opra(7));
        elsif (s_sbrcs='1') then
                s_statsbics <= '0';s_statsbrcs <= '1';s_statcpse <= '0';s_stat32 <= '0';
                s_statst <= '0';s_statst2 <= '0';s_statget <= '0';s_statstore <= '0';
                s_statlpm <= '0';s_statspm0 <= '0';s_statspm <= '0';
                s_statspm1 <= '0';s_statspm2 <= '0';state <= sbrcsstate;
        elsif (s_rcall='1') then
                stack7 <= stack6;stack6 <= stack5;stack5 <= stack4;stack4 <= stack3;
                stack3 <= stack2;stack2 <= stack1;stack1 <= stack0;stack0 <= tpstack;
        elsif (s_icall='1') then
                stack7 <= stack6;stack6 <= stack5;stack5 <= stack4;stack4 <= stack3;
                stack3 <= stack2;stack2 <= stack1;stack1 <= stack0;stack0 <= tpstack;
        elsif (s_ret='1' or s_reti='1') then
                stack6 <= stack7;stack5 <= stack6;stack4 <= stack5;stack3 <= stack4;
                stack2 <= stack3;stack1 <= stack2;stack0 <= stack1;
        elsif (s_push='1') then
                stack7 <= stack6;stack6 <= stack5;stack5 <= stack4;stack4 <= stack3;
                stack3 <= stack2;stack2 <= stack1;stack1 <= stack0;
                stack0 <= '0' & gpr(conv_integer(pin_instir(7 downto 4)));
        elsif (s_pop='1') then
                gpr(conv_integer(pin_instir(7 downto 4))) <= stack0(7 downto 0);
                stack6 <= stack7;stack5 <= stack6;stack4 <= stack5;stack3 <= stack4;
                stack2 <= stack3;stack1 <= stack2;stack0 <= stack1;
        elsif (s_spm='1') then
                s_statsbics <= '0';s_statsbrcs <= '0';s_statcpse <= '0';
                s_stat32 <= '0';s_statst <= '0';s_statst2 <= '0';s_statst3 <= '0';
                s_statget <= '0';s_statstore <= '0';s_statlpm <= '0';s_statspm0 <= '1';
                s_statspm <= '0';s_statspm1 <= '0';s_statspm2 <= '0';pin_werom <= '0';
                state <= spmstore1;
        else
        end if;
    when standbystate =>
        if (s_int0int='1' or pin_tov='1') then
                s_statsbics <= '0';s_statsbrcs <= '0';s_statcpse <= '0';s_stat32 <= '0';
                s_statst <= '0';s_statst2 <= '0';s_statst3 <= '0';s_statget <= '0';
                s_statstore <= '0';s_statlpm <= '0';s_statspm0 <= '0';s_statspm <= '0';
                s_statspm1 <= '0';s_statspm2 <= '0';state <= exestate;
        else
                s_statsbics <= '1';s_statsbrcs <= '1';s_statcpse <= '1';s_stat32 <= '1';
                s_statst <= '1';s_statst2 <= '1';s_statst3 <= '1';s_statget <= '1';
                s_statstore <= '1';s_statlpm <= '1';s_statspm0 <= '1';s_statspm <= '1';
                s_statspm1 <= '1';s_statspm2 <= '1';state <= standbystate;
        end if;
```

```
when exestate32 =>
      gpr(conv_integer(s_regd32inst)) <= s_outram;
      s_statsbics <= '0';s_statsbrcs <= '0';s_statcpse <= '0';s_stat32 <= '0';
      s_statst <= '0';s_statst2 <= '0';s_statst3 <= '0';s_statget <= '0';s_statstore <= '0';
      s_statlpm <= '0';s_statspm0 <= '0';s_statspm <= '0';s_statspm1 <= '0';
      s_statspm2 <= '0';state <= exestate;
when store2 =>
      s_statsbics <= '0';s_statsbrcs <= '0';s_statcpse <= '0';s_stat32 <= '0';
      s_statst <= '0';s_statst2 <= '1';s_statst3 <= '0';s_statget <= '0';s_statstore <= '0';
      s_statlpm <= '0';s_statspm0 <= '0';s_statspm <= '0';s_statspm1 <= '0';
      s_statspm2 <= '0';s_enastclk <= '0';state <= store3;
when store3 =>
      s_statsbics <= '0';s_statsbrcs <= '0';s_statcpse <= '0';s_stat32 <= '0';
      s_statst <= '0';s_statst2 <= '0';s_statst3 <= '1';s_statget <= '0';s_statstore <= '0';
      s_statlpm <= '0';s_statspm0 <= '0';s_statspm <= '0';s_statspm1 <= '0';
      s_statspm2 <= '0';state <= store4;
when store4 =>
      s_statsbics <= '0';s_statsbrcs <= '0';s_statcpse <= '0';s_stat32 <= '0';
      s_statst <= '0';s_statst2 <= '0';s_statst3 <= '0';s_statget <= '0';s_statstore <= '0';
      s_statlpm <= '0';s_statspm0 <= '0';s_statspm <= '0';s_statspm1 <= '0';
      s_statspm2 <= '0';state <= exestate;
when getaddress =>
      s_statsbics <= '0';s_statsbrcs <= '0';s_statcpse <= '0';s_stat32 <= '0';
      s_statst <= '0';s_statst2 <= '0';s_statst3 <= '0';s_statget <= '0';s_statstore <= '1';
      s_statlpm <= '0';s_statspm0 <= '0';s_statspm <= '0';s_statspm1 <= '0';
      s_statspm2 <= '0';state <= store2sts;
when store2sts =>
      s_statsbics <= '0';s_statsbrcs <= '0';s_statcpse <= '0';s_stat32 <= '0';
      s_statst <= '0';s_statst2 <= '0';s_statst3 <= '0';s_statget <= '0';s_statstore <= '0';
      s_statlpm <= '0';s_statspm0 <= '0';s_statspm <= '0';s_statspm1 <= '0';
      s_statspm2 <= '0';state <= exestate;
when lpmstate =>
      s_statsbics <= '0';s_statsbrcs <= '0';s_statcpse <= '0';s_stat32 <= '0';
      s_statst <= '0';s_statst2 <= '0';s_statst3 <= '0';s_statget <= '0';s_statstore <= '0';
      s_statlpm <= '0';s_statspm0 <= '0';s_statspm <= '0';s_statspm1 <= '0';
      s_statspm2 <= '0';state <= exestate;
      gpr(conv_integer(s_regd32inst)) <= s_instir;
      if s_zinclpm='1' then
            gpr(14) <= s_dupz + 1;
      end if;
when cpsestate =>
      s_statsbics <= '0';s_statsbrcs <= '0';s_statcpse <= '0';s_stat32 <= '0';
      s_statst <= '0';s_statst2 <= '0';s_statst3 <= '0';s_statget <= '0';s_statstore <= '0';
      s_statlpm <= '0';s_statspm0 <= '0';s_statspm <= '0';s_statspm1 <= '0';
      s_statspm2 <= '0';state <= exestate;
when sbrcsstate =>
      s_statsbics <= '0';s_statsbrcs <= '0';s_statcpse <= '0';s_stat32 <= '0';
      s_statst <= '0';s_statst2 <= '0';s_statst3 <= '0';s_statget <= '0';s_statstore <= '0';
      s_statlpm <= '0';s_statspm0 <= '0';s_statspm <= '0';s_statspm1 <= '0';
      s_statspm2 <= '0';state <= exestate;
when sbicsstate =>
      s_statsbics <= '0';s_statsbrcs <= '0';s_statcpse <= '0';s_stat32 <= '0';
      s_statst <= '0';s_statst2 <= '0';s_statst3 <= '0';s_statget <= '0';s_statstore <= '0';
      s_statlpm <= '0';s_statspm0 <= '0';s_statspm <= '0';s_statspm1 <= '0';
      s_statspm2 <= '0';state <= exestate;
when spmstore1 =>
      s_statsbics <= '0';s_statsbrcs <= '0';s_statcpse <= '0';s_stat32 <= '0';
      s_statst <= '0';s_statst2 <= '0';s_statst3 <= '0';s_statget <= '0';s_statstore <= '0';
      s_statlpm <= '0';s_statspm0 <= '0';s_statspm <= '1';s_statspm1 <= '0';
      s_statspm2 <= '0';pin_werom <= '1';state <= spmstore2;
when spmstore2 =>
      s_statsbics <= '0';s_statsbrcs <= '0';s_statcpse <= '0';s_stat32 <= '0';
      s_statst <= '0';s_statst2 <= '0';s_statst3 <= '0';s_statget <= '0';s_statstore <= '0';
      s_statlpm <= '0';s_statspm0 <= '0';s_statspm <= '0';s_statspm1 <= '1';
      s_statspm2 <= '0';pin_werom <= '0';state <= spmstore3;
when spmstore3 =>
      s_statsbics <= '0';s_statsbrcs <= '0';s_statcpse <= '0';s_stat32 <= '0';
      s_statst <= '0';s_statst2 <= '0';s_statst3 <= '0';s_statget <= '0';s_statstore <= '0';
      s_statlpm <= '0';s_statspm0 <= '0';s_statspm <= '0';s_statspm1 <= '0';
```

```
        s_statspm2 <= '1';pin_werom <= '0';state <= spmstore4;
    when spmstore4 =>
        s_statsbics <= '0';s_statsbrcs <= '0';s_statcpse <= '0';s_stat32 <= '0';
        s_statst <= '0';s_statst2 <= '0';s_statst3 <= '0';s_statget <= '0';s_statstore <= '0';
        s_statlpm <= '0';s_statspm0 <= '0';s_statspm <= '0';s_statspm1 <= '0';
        s_statspm2 <= '0';state <= exestate;
    when others =>
  end case;
end if;
end process;
ena_extirq <= s_gicr(6) and sr(7); -- ini keluar ke enable external unit
pin_toie <= s_timsk(0) and sr(7);pin_cso <= s_tccr(2 downto 0);s_timerint <= pin_tov;
pin_wde <= s_wdtcr(3);pin_wdp <= s_wdtcr(2 downto 0);
end Arch_CoreUnit;
```

***Source code* PCUnit.vhd**

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity PCUnit is
port(     pin_clk,pin_rst : in std_logic;
          pin_enable : in std_logic;
          pin_pc : buffer std_logic_vector(8 downto 0);
          pin_addr : in std_logic_vector(8 downto 0);
          pin_indirect : in std_logic;
          pin_int0int,pin_timerint : in std_logic);
end PCUnit;
architecture Arch_PCUnit of PCUnit is
constant int_int0 : std_logic_vector(8 downto 0) := "000000001";
constant int_timer : std_logic_vector(8 downto 0) := "000000010";
begin
process(pin_clk,pin_rst)
begin
if pin_rst='0' then
  pin_pc <= "000000000";
elsif pin_clk'event and pin_clk='0' then
  if pin_enable='1' then
          if pin_int0int='1' then
                    pin_pc <= int_int0;
          elsif pin_timerint='1' then
                    pin_pc <= int_timer;
          elsif pin_indirect='1' then
                    pin_pc <= pin_addr;
          else
                    pin_pc <= pin_pc + pin_addr + 1;
          end if;
  end if;
end if;
end process;
end Arch_PCUnit;
```

***Source code* RAMUnit.vhd**

```
library ieee;
use ieee.std_logic_1164.all;
library lpm;
use lpm.lpm_components.all;
entity ramunit is
          port(     address          : in std_logic_vector (7 downto 0);
                    we               : in std_logic  := '1';
                    data             : in std_logic_vector (7 downto 0);
                    q                : out std_logic_vector (7 downto 0));
end ramunit;
architecture arch_ramunit of ramunit is
          signal sub_wire0  : std_logic_vector (7 downto 0);
```

```
        component lpm_ram_dq
        generic (lpm_width              : natural;
                 lpm_widthad      : natural;
                 lpm_indata             : string;
                 lpm_address_control    : string;
                 lpm_outdata            : string;
                 lpm_file           : string;
                 lpm_hint           : string);
        port (   address  : in std_logic_vector (7 downto 0);
                 q        : out std_logic_vector (7 downto 0);
                 data     : in std_logic_vector (7 downto 0);
                 we       : in std_logic);
        end component;
begin
        q    <= sub_wire0(7 downto 0);
        lpm_ram_dq_component : lpm_ram_dq
        generic map (lpm_width => 8,lpm_widthad => 8,lpm_indata => "unregistered",
                 lpm_address_control => "unregistered",lpm_outdata => "unregistered",
                 lpm_hint => "use_eab=on")
        port map (address => address,data => data,we => we,q => sub_wire0);
end arch_ramunit;
```

Source code buffExtInt.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity buffExtInt is
port(pin_enable : in std_logic;
        pin_in      : in std_logic;
        pin_out     : out std_logic);
end buffExtInt;
architecture Arch_buffExtInt of buffExtInt is
begin
process(pin_enable)
begin
case pin_enable is
when '1' => pin_out <= pin_in;
when others => pin_out <= '0';
end case;
end process;
end Arch_buffExtInt;
```

Source code ShifterUnit.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity ShifterUnit is
port(    pin_enable : in std_logic;
        pin_in : in std_logic_vector(7 downto 0);
        pin_selop : in integer range 0 to 2;
        pin_cflagin : in std_logic;
        pin_cflagout : out std_logic;
        pin_out : out std_logic_vector(7 downto 0));
end ShifterUnit;
architecture Arch_ShifterUnit of ShifterUnit is
begin
process(pin_enable,pin_in)
begin
if pin_enable='1' then
 if pin_selop=0 then                -- lsr -> bit 7:0,bit 0 masuk C-Flag
        pin_out(7) <= '0';
 elsif pin_selop=1 then             -- ror -> bit 7:C-Flag,bit 0 masuk C-Flag
        pin_out(7) <= pin_cflagin;
 elsif pin_selop=2 then             -- asr -> bit 7:bit 7,bit 0 masuk C-Flag
        pin_out(7) <= pin_in(7);
 end if;
```

```
 pin_out(6 downto 0) <= pin_in(7 downto 1);
 pin_cflagout <= pin_in(0);
end if;
end process;
end Arch_ShifterUnit;
```

***Source code* SwapUnit.vhd**

```
library ieee;
use ieee.std_logic_1164.all;

entity SwapUnit is
port(     pin_in : in std_logic_vector(7 downto 0);
          pin_out : out std_logic_vector(7 downto 0));
end SwapUnit;
architecture Arch_SwapUnit of SwapUnit is
begin
process(pin_in)
begin
          pin_out(3 downto 0) <= pin_in(7 downto 4);
          pin_out(7 downto 4) <= pin_in(3 downto 0);
end process;
end Arch_SwapUnit;
```

***Source code* LogicUnit.vhd**

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity LogicUnit is
port(pin_enable : in std_logic;
   pin_opra,pin_oprb : in std_logic_vector(7 downto 0);
   pin_selop : in integer range 0 to 3;
   pin_out : out std_logic_vector(7 downto 0));
end LogicUnit;
architecture Arch_LogicUnit of LogicUnit is
begin
process(pin_enable,pin_opra,pin_oprb)
begin
if pin_enable='1' then
 if pin_selop=0 then                                    -- and,andi
          pin_out <= pin_opra and pin_oprb;
 elsif pin_selop=1 then                                 -- or,ori
          pin_out <= pin_opra or pin_oprb;
 elsif pin_selop=2 then                                 -- eor
          pin_out <= pin_opra xor pin_oprb;
 elsif pin_selop=3 then                                 -- com
          pin_out <= not pin_opra;
 end if;
end if;
end process;
end Arch_LogicUnit;
```

***Source code* MultUnit.vhd**

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity MultUnit is
port (    a: in std_logic_vector (7 downto 0);
          b: in std_logic_vector (7 downto 0);
          sign : in std_logic;
          shift : in std_logic;
          result_high: out std_logic_vector (7 downto 0);
          result_low: out std_logic_vector (7 downto 0);
```

```
        pin_carry : out std_logic);
end MultUnit;
architecture Arch_MultUnit of MultUnit is
signal s_a,s_b : std_logic_vector(7 downto 0);
signal s_result,s_final,s_resfinal : std_logic_vector(15 downto 0);
begin
process(sign,a,b)
begin
if (sign='0') then              -- MUL,FMUL
        s_a <= a;s_b <= b;
else                            -- MULS,FMULS
        if a(7)='1' then
                s_a <= not a + 1;
        else
                s_a <= a;
        end if;
        if b(7)='1' then
                s_b <= not b + 1;
        else
                s_b <= b;
        end if;
end if;
end process;
s_result <= s_a * s_b;
process(s_result)
begin
        if sign='1' then
                if a(7)/=b(7) then
                        s_final <= not s_result + 1;
                else
                        s_final <= s_result;
                end if;
        else
                s_final <= s_result;
        end if;
end process;
process(s_final)
begin
if shift='1' then
        s_resfinal(15 downto 1) <= s_final(14 downto 0);
        pin_carry <= s_final(15);
        s_resfinal(0) <= '0';
else
        pin_carry <= '0';
        s_resfinal <= s_final;
end if;
end process;
result_high <= s_resfinal(15 downto 8);
result_low <= s_resfinal(7 downto 0);
end Arch_MultUnit;
```

***Source code* MulsuUnit.vhd**

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity MulsuUnit is
port(   a,b : in std_logic_vector(7 downto 0);
        shift : in std_logic;
        result_high: out std_logic_vector (7 downto 0);
        result_low: out std_logic_vector (7 downto 0);
        pin_carry : out std_logic);
end MulsuUnit;
architecture Arch_MulsuUnit of MulsuUnit is
signal s_a,s_b : std_logic_vector(7 downto 0);
signal s_result,s_resfinal,s_resultfin : std_logic_vector(15 downto 0);
```

```
signal sign_a : std_logic;
begin
process(a)
begin
if a(7)='1' then
  s_a <= not a + 1;sign_a <= '1';
else
  s_a <= a;sign_a <= '0';
end if;
end process;
s_b <= b;
s_result <= s_a * s_b;
process(s_result,shift)
begin
if shift='1' then
        s_resfinal(15 downto 1) <= s_result(14 downto 0);
        pin_carry <= s_result(15);s_resfinal(0) <= '0';
else
        pin_carry <= '0';s_resfinal <= s_result;
end if;
end process;
process(s_resfinal,sign_a)
begin
   if sign_a='1' then s_resultfin <= not s_resfinal + 1;
   else s_resultfin <= s_resfinal; end if;
end process;
result_high <= s_resultfin(15 downto 8);
result_low <= s_resultfin(7 downto 0);
end Arch_MulsuUnit;
```

***Source code* AddHighUnit.vhd**

```
library ieee;
use ieee.std_logic_1164.all;
library lpm;
use lpm.lpm_components.all;
entity AddHighUnit is
port(   pin_opra,pin_oprb : in std_logic_vector(7 downto 0);
        pin_cin,pin_addsub : in std_logic;
        pin_result : out std_logic_vector(7 downto 0);
        pin_cout,pin_overflow : out std_logic);
end AddHighUnit;
architecture Arch_AddHighUnit of AddHighUnit is
begin
addpart: lpm_add_sub generic map (lpm_width=>8) port map
(dataa=>pin_opra,datab=>pin_oprb,cin=>pin_cin,add_sub=>pin_addsub,result=>pin_result,
cout=>pin_cout,overflow=>pin_overflow);
end Arch_AddHighUnit;
```

***Source code* AddLowUnit.vhd**

```
library ieee;
use ieee.std_logic_1164.all;
library lpm;
use lpm.lpm_components.all;
entity AddLowUnit is
port(   pin_opra,pin_oprb : in std_logic_vector(7 downto 0);
        pin_cin,pin_addsub : in std_logic;
        pin_result : out std_logic_vector(7 downto 0);
        pin_cout,pin_overflow : out std_logic);
end AddLowUnit;
architecture Arch_AddLowUnit of AddLowUnit is
begin
addpart: lpm_add_sub generic map (lpm_width=>8) port map
(dataa=>pin_opra,datab=>pin_oprb,cin=>pin_cin,add_sub=>pin_addsub,result=>pin_result,
cout=>pin_cout,overflow=>pin_overflow);
end Arch_AddLowUnit;
```

Source code TimerInterruptUnit.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity TimerInterruptUnit is
port(     pin_clk,pin_rst : in std_logic;
          pin_toie,pin_wrtcnt : in std_logic;
          pin_tcnt : in std_logic_vector(7 downto 0);
          pin_cso : in std_logic_vector(2 downto 0);
          pin_Binport7 : in std_logic;
          pin_tov : out std_logic);
end TimerInterruptUnit;
architecture Arch_TimerInterruptUnit of TimerInterruptUnit is
component prescaller
port(     pin_clk,pin_rst : in std_logic;
          pin_cso : in std_logic_vector(2 downto 0);
          pin_extport : in std_logic;
          pin_timerclk : out std_logic);
end component;
signal s_clktimer : std_logic;
component TimerUnit
port(     pin_clk,pin_rst : in std_logic;
          pin_toie : in std_logic;
          pin_wrtcnt : in std_logic;
          pin_tcnt : in std_logic_vector(7 downto 0);
          pin_tov : out std_logic);
end component;
begin
prescallerPart: prescaller port map (pin_clk,pin_rst,pin_cso,pin_Binport7,s_clktimer);
TimerUnitPart: TimerUnit port map (s_clktimer,pin_rst,pin_toie,pin_wrtcnt,pin_tcnt,pin_tov);
end Arch_TimerInterruptUnit;
```

Source code Prescaller.vhd

```
library ieee;
use ieee.std_logic_1164.all;
entity prescaller is
port(     pin_clk,pin_rst : in std_logic;
          pin_cso : in std_logic_vector(2 downto 0);
          pin_extport : in std_logic;
          pin_timerclk : out std_logic);
end prescaller;
architecture Arch_prescaller of prescaller is
signal s_div2,s_div4,s_div8,s_div16,s_div32,s_div64,s_div128,s_div256,
          s_div512,s_div1024 : std_logic;
begin
process(pin_clk,pin_rst)
begin
if (pin_rst='0') then
          s_div2 <= '0';  s_div4 <= '0'; s_div8 <= '0'; s_div16 <= '0'; s_div32 <= '0';
          s_div64 <= '0'; s_div128 <= '0'; s_div256 <= '0'; s_div512 <= '0'; s_div1024 <= '0';
elsif (pin_clk'event and pin_clk='1') then
  s_div2 <= not s_div2;
  if s_div2='1' then
  s_div4 <= not s_div4;
          if s_div4='1' then
          s_div8 <= not s_div8;
                    if s_div8='1' then
                    s_div16 <= not s_div16;
                              if s_div16='1' then
                              s_div32 <= not s_div32;
                                        if s_div32='1' then
                                        s_div64  <= not s_div64;
                                                  if s_div64='1' then
                                                  s_div128 <= not s_div128;
                                                            if s_div128='1' then
                                                            s_div256 <= not s_div256;
```

```
                                                    if s_div256='1' then
                                                    s_div512 <= not s_div512;
                                                            if s_div512='1' then
                                                    s_div1024 <= not s_div1024;
                                                            end if;
                                                    end if;
                                            end if;
                                    end if;
                            end if;
                    end if;
            end if;
    end if;
end if;
end process;
process(pin_cso,pin_clk,s_div8,s_div64,s_div256,s_div1024,pin_extport)
begin
case pin_cso is
   when "000" => pin_timerclk <= '0';                -- off
   when "001" => pin_timerclk <= pin_clk;            -- clock
   when "010" => pin_timerclk <= s_div8;             -- clock/8
   when "011" => pin_timerclk <= s_div64;            -- clock/64
   when "100" => pin_timerclk <= s_div256;           -- clock/256
   when "101" => pin_timerclk <= s_div1024;          -- clock/1024
   when "110" => pin_timerclk <= not pin_extport; -- external pin,NGT
   when "111" => pin_timerclk <= pin_extport;        -- external pin,PGT
   when others =>
end case;
end process;
end Arch_prescaller;
```

***Source code* TimerUnit.vhd**

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity TimerUnit is
port(     pin_clk,pin_rst : in std_logic;
          pin_toie : in std_logic;
          pin_wrtcnt : in std_logic;
          pin_tcnt : in std_logic_vector(7 downto 0);
          pin_tov : out std_logic);
end TimerUnit;
architecture Arch_TimerUnit of TimerUnit is
signal s_tcnt : std_logic_vector(7 downto 0);
begin
process(pin_clk,pin_rst)
begin
if pin_rst='0' then
          s_tcnt <= "00000000";
elsif pin_clk'event and pin_clk='1' then
   if (pin_wrtcnt='1' and pin_toie='0') then
          s_tcnt <= pin_tcnt;
   elsif (pin_wrtcnt='0' and pin_toie='1') then
          s_tcnt <= s_tcnt + 1;
          if s_tcnt="11111111" then pin_tov <= '1';end if;
   else
          pin_tov <= '0';
   end if;
end if;
end process;
end Arch_TimerUnit;
```

***Source code* PrescallerWDR.vhd**

```
library ieee;
use ieee.std_logic_1164.all;
entity prescallerWDR is
```

```
port(pin_clk,pin_rst : in std_logic;
      pin_wde : in std_logic;
      pin_wdp : in std_logic_vector(2 downto 0);
      pin_wdrrst : out std_logic);
end prescallerWDR;
architecture Arch_prescallerWDR of prescallerWDR is
signal s_div2,s_div4,s_div8,s_div16,s_div32,s_div64,s_div128,s_div256,
s_div512,s_div1024,s_div2048 : std_logic;
begin
process(pin_clk,pin_rst)
begin
if (pin_rst='0') then
s_div2 <= '0'; s_div4 <= '0'; s_div8 <= '0'; s_div16 <= '0'; s_div32 <= '0';s_div64 <= '0';
s_div128 <= '0'; s_div256 <= '0'; s_div512 <= '0'; s_div1024 <= '0'; s_div2048 <= '0';
elsif (pin_clk'event and pin_clk='1') then
 if pin_wde='1' then
 s_div2 <= not s_div2;
   if s_div2='1' then
   s_div4 <= not s_div4;
          if s_div4='1' then
          s_div8 <= not s_div8;
                   if s_div8='1' then
                   s_div16 <= not s_div16;
                            if s_div16='1' then
                            s_div32 <= not s_div32;
                                     if s_div32='1' then
                                     s_div64  <= not s_div64;
                                              if s_div64='1' then
                                              s_div128 <= not s_div128;
                                                       if s_div128='1' then
                                                       s_div256 <= not s_div256;
                                                                if s_div256='1' then
                                                                s_div512 <= not s_div512;
                                                                         if s_div512='1' then
                                                                        s_div1024 <= not div1024;
                                                                         if s_div1024='1' then
                                                                s_div2048 <= not s_div1024;
                                                                                 end if;
                                                                         end if;
                                                                end if;
                                                       end if;
                                              end if;
                                     end if;
                            end if;
                   end if;
          end if;
   end if;
 else
    s_div2 <= '0'; s_div4 <= '0'; s_div8 <= '0'; s_div16 <= '0'; s_div32 <= '0'; s_div64 <= '0';
    s_div128 <= '0'; s_div256 <= '0'; s_div512 <= '0'; s_div1024 <= '0'; s_div2048 <= '0';
 end if;
end if;
end process;
process(pin_wdp,s_div16,s_div32,s_div64,s_div128,s_div256,s_div512,s_div1024,s_div2048)
begin
case pin_wdp is
   when "000" => pin_wdrrst <= s_div16;          -- clock/16
   when "001" => pin_wdrrst <= s_div32;          -- clock/32
   when "010" => pin_wdrrst <= s_div64;          -- clock/64
   when "011" => pin_wdrrst <= s_div128;         -- clock/128
   when "100" => pin_wdrrst <= s_div256;         -- clock/256
   when "101" => pin_wdrrst <= s_div512;         -- clock/512
   when "110" => pin_wdrrst <= s_div1024;        -- clock/1024
   when "111" => pin_wdrrst <= s_div2048;        -- clock/2048
   when others =>
end case;
end process;
end Arch_prescallerWDR;
```