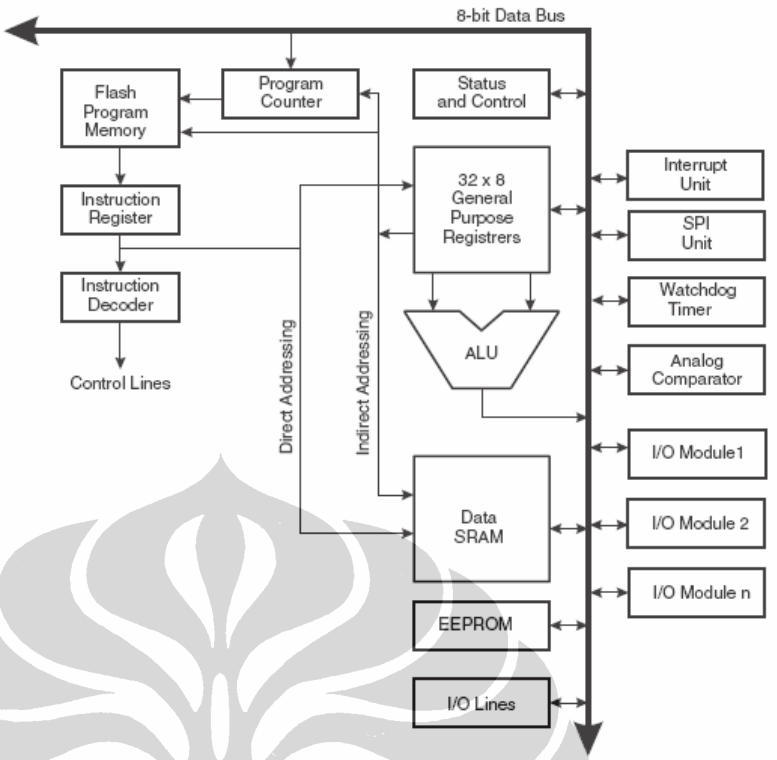
BAB II
PENGENALAN ATMEGA 8535 DAN VHDL

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].

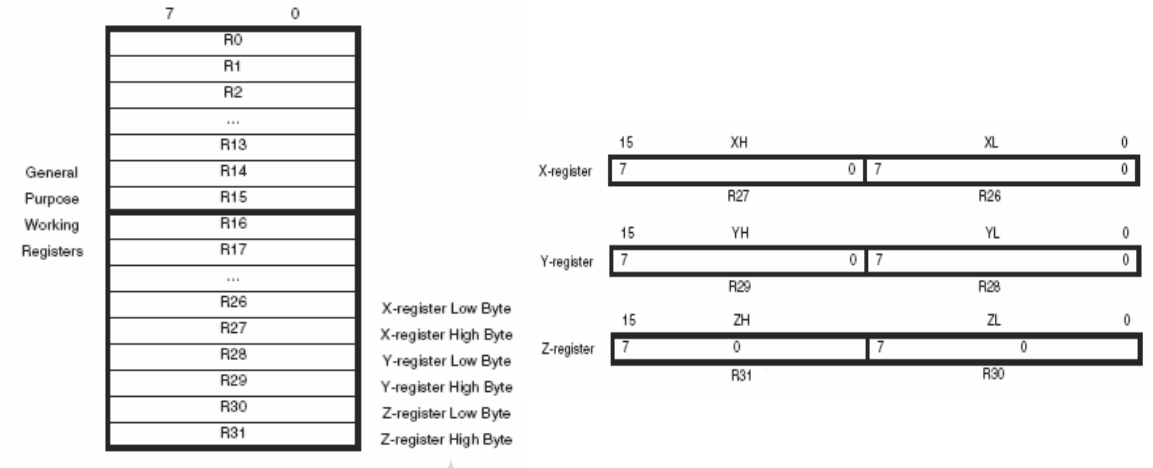
Diagram blok arsitektur dari mikrokontroler Atmel ATMega 8535 secara umum diperlihatkan pada Gambar 2.1. Arsitektur dari mikrokontroler ini secara umum dibuat sedemikian agar mikrokontroler dapat melakukan pengaksesan *memory*, melaksanakan perhitungan, mengontrol modul-modul yang ada, dan menangani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



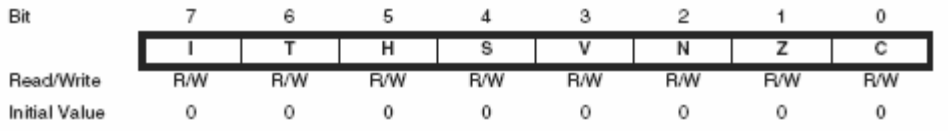
(a) (b)

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:

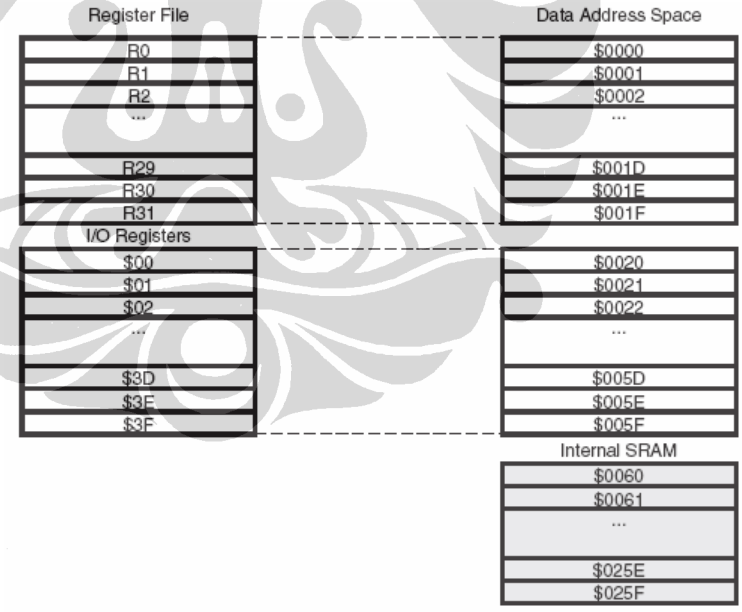
1. I (*Global Interrupt Enable*) digunakan untuk mengaktifkan fungsi *interrupt*. *Flag* ini harus berlogika '1' untuk mengaktifkan fungsi *interrupt* dan sebaliknya. Fungsi *enable* dari masing-masing *interrupt* diatur tersendiri.
2. T (*Bit Copy Storage*). Instruksi BLD (*Bit Load*) dan BST (*Bit 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]

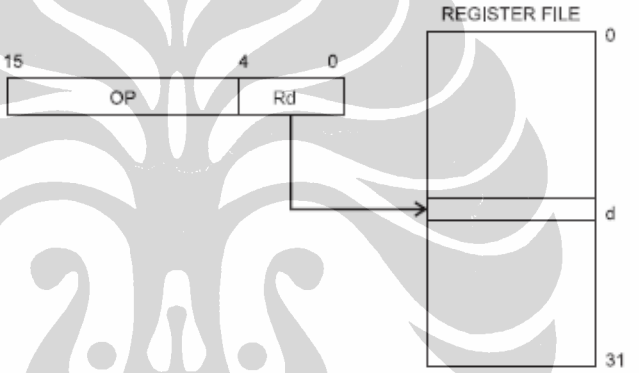
Saat terjadi *interrupt* dan pemanggilan *subroutine*, alamat instruksi selanjutnya dari *Program Counter* akan disimpan sementara di dalam *stack* yang berada di dalam SRAM. Program akan kembali pada alamat yang tersimpan pada

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*)

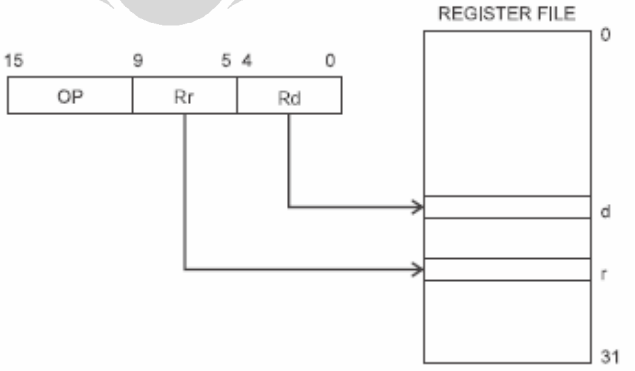
Mikrokontroler Atmel ATMega 8535 mendukung pengalamatan untuk mengakses *program memory* (*Flash*) dan *data memory* (SRAM, *register file*, *I/O memory*). Jenis pengalamatan yang didukung ada 12 buah jenis pengalamatan, yaitu:

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



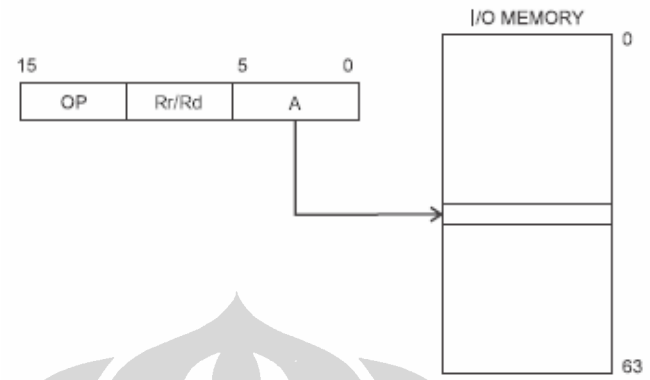
Gambar 2.5 Pengalamatan *register direct, single register Rd* [6]

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*.



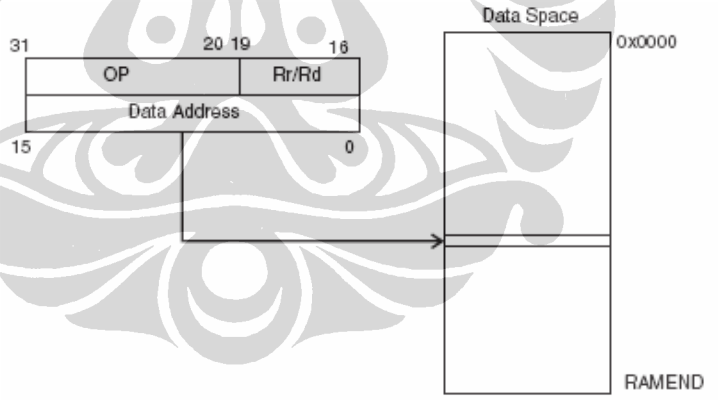
Gambar 2.6 Pengalamatan *register direct, two register Rd-Rr* [6]

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.



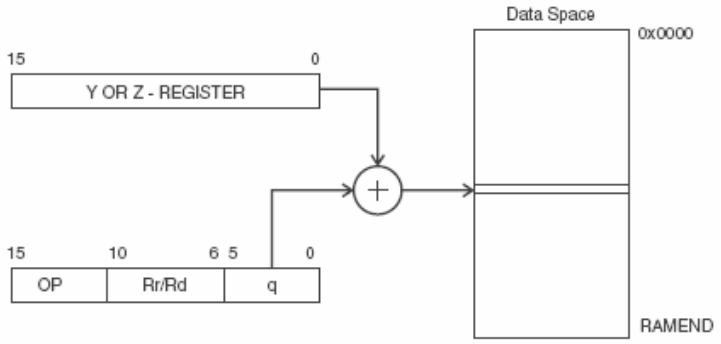
Gambar 2.7 Pengalamatan *I/O direct* [6]

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.



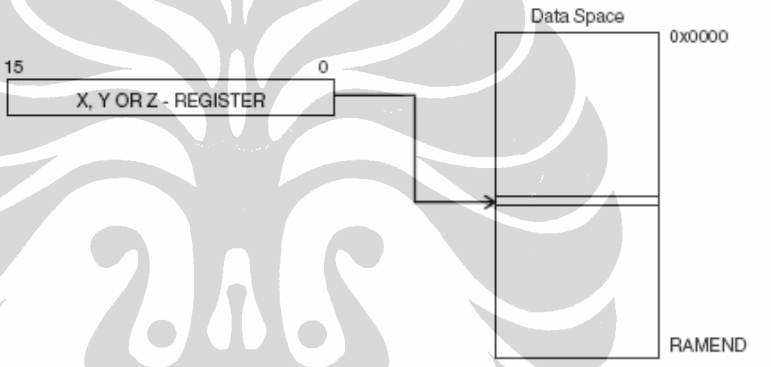
Gambar 2.8 Pengalamatan *relative program* [6]

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*.



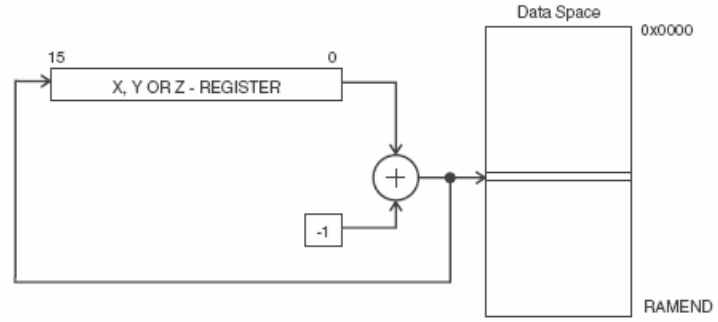
Gambar 2.9 Pengalamatan *data indirect with displacement* [6]

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



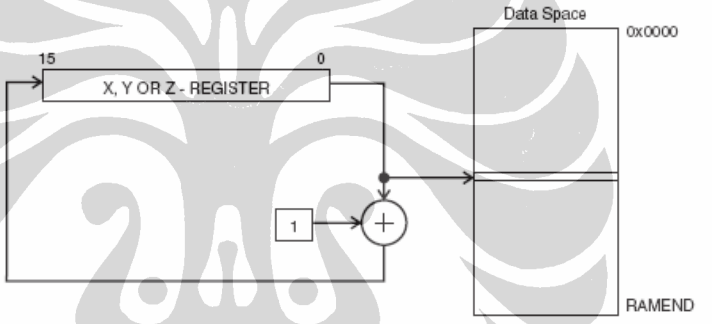
Gambar 2.10 Pengalamatan *data indirect* [6]

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.



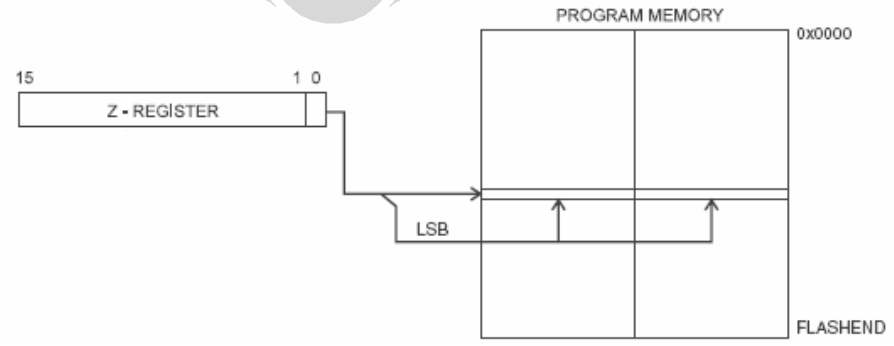
Gambar 2.11 Pengalamatan *data indirect with pre-decrement* [6]

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.



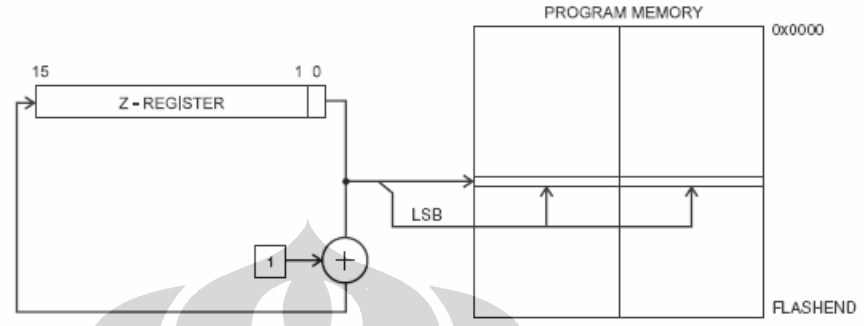
Gambar 2.12 Pengalamatan *data indirect with post-increment* [6]

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



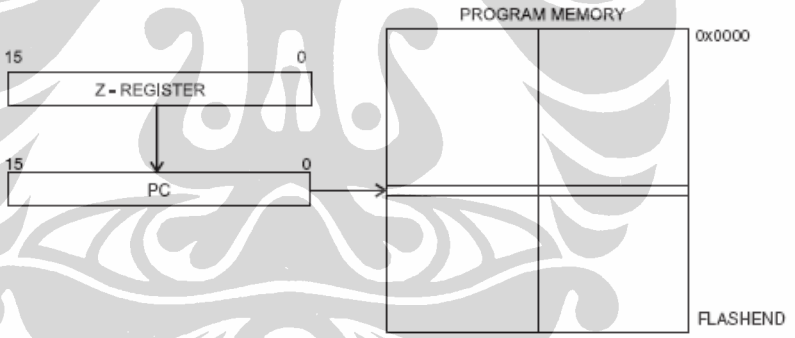
Gambar 2.13 Pengalamatan *program memory constant* [6]

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.



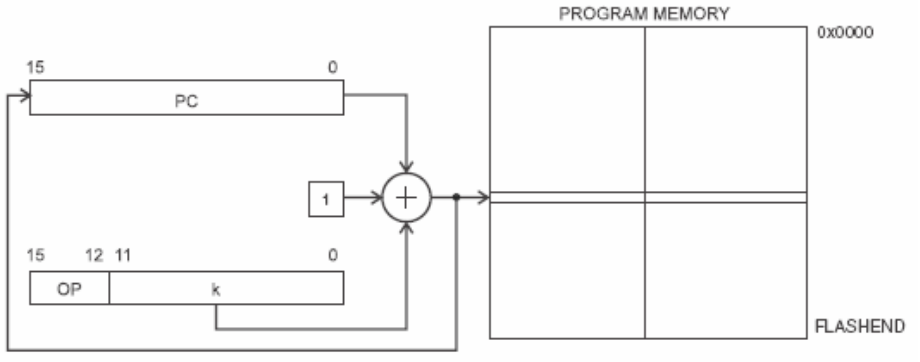
Gambar 2.14 pengalamatan *program memory with post-increment* [6]

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.

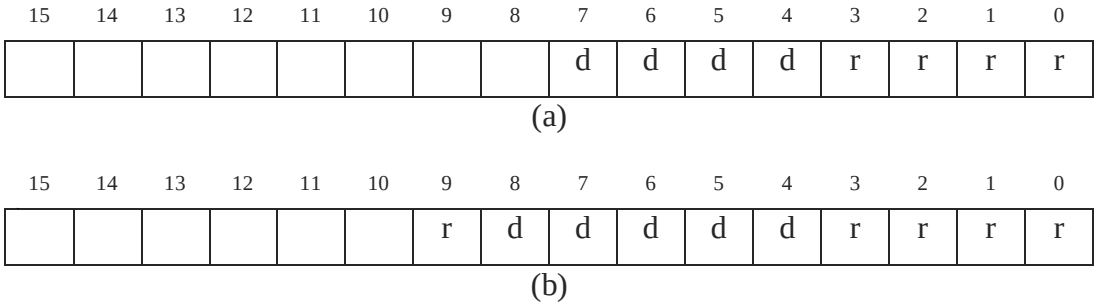
Instruksi bit dan tes bit yang dimiliki oleh ATMega 8535 berjumlah 28 buah instruksi. Kelompok instruksi ini akan mempengaruhi nilai *flag* yang tersimpan di *Status Register*, kecuali instruksi SBI (*Set Bit in I/O Register*), CBI (*Clear Bit in I/O Register*), SWAP (*Swap Nibbles*), dan BLD (*Bit Load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

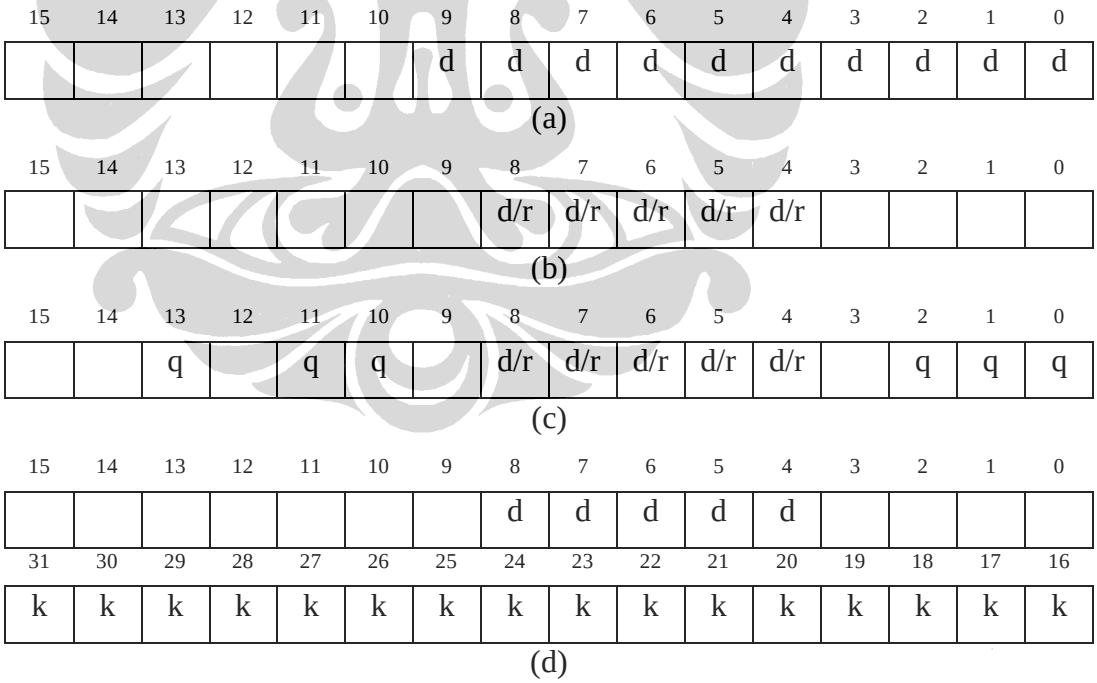
Mikrokontroler Atmel ATMega 8535 memiliki panjang kode instruksi yang tetap, yaitu sebesar 16-bit dan 32-bit dengan pola tertentu. Instruksi-instruksi ini dapat dikelompokkan menjadi 9 kelompok, yaitu pola instruksi yang membutuhkan dua buah register, sebuah register, instruksi untuk operasi yang melibatkan 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.



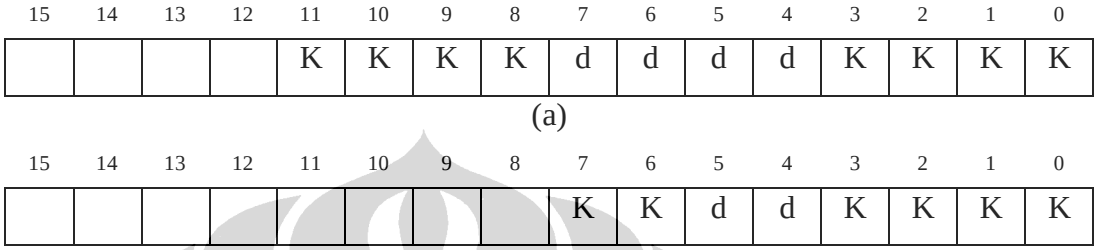
Gambar 2.17 Pola instruksi dengan 2 buah *register* (a) pola 1 (b) pola 2

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.



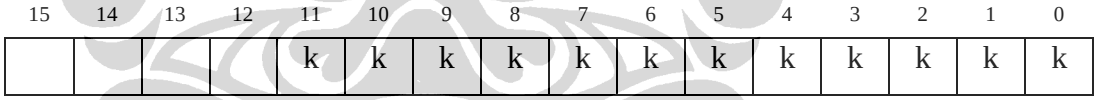
Gambar 2.18 Pola instruksi dengan panjang 16-*bit* (a) pola 1 (b) pola 2 (c) pola 3, (d) pola instruksi dengan panjang 32-*bit*

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.



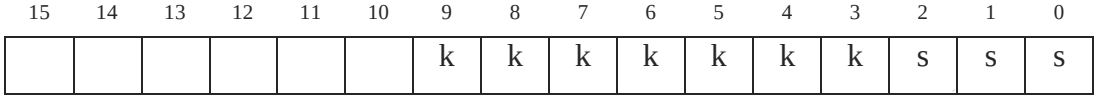
Gambar 2.19 Pola instruksi yang melibatkan nilai konstanta tertentu (a) pola 1 (b) pola 2

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*.



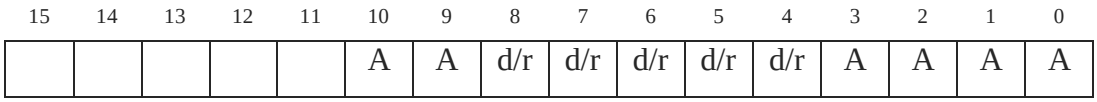
Gambar 2.20 Pola instruksi untuk percabangan tidak bersyarat

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.



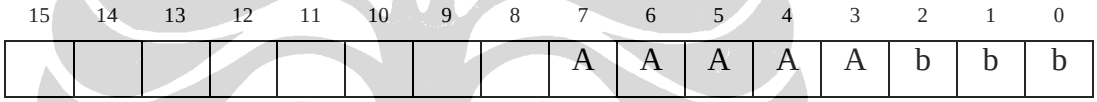
Gambar 2.21 Pola instruksi untuk percabangan bersyarat

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*.



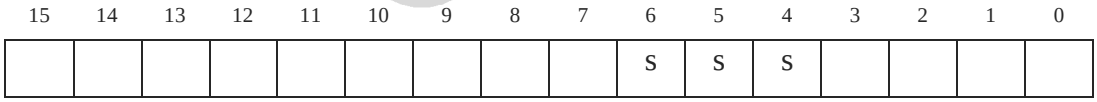
Gambar 2.22 Pola instruksi yang melibatkan 64 alamat I/O *register*

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.



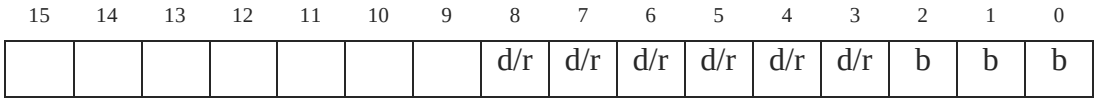
Gambar 2.23 Pola instruksi yang melibatkan 32 alamat terbawah I/O *register*

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)

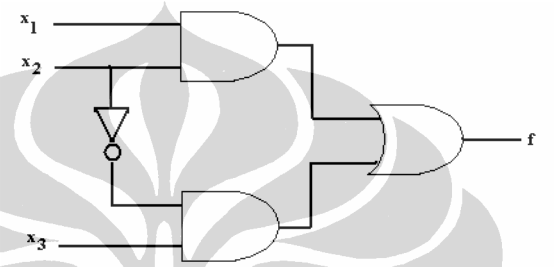
Sebuah sistem dijital dapat dideskripsikan sebagai proses perpindahan *register* dengan menggunakan bahasa deskripsi perangkat keras [7]. *Very High Speed Integrated Circuit* (VHSIC) dikembangkan dalam rangka kebutuhan para perancang rangkaian dijital yang membutuhkan sarana untuk mendeskripsikan struktur 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*.