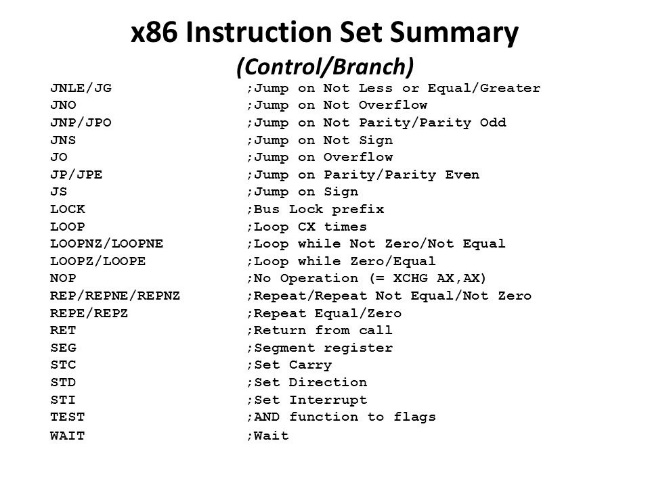
**TUGAS 1**

**INSTRUCTION SET x86  
ORGANISASI DAN ARSITEKTUR KOMPUTER**

Pengantar :

Instruction Set x86 Family

****

X86 Intruction Set Summary

**Dosen Pengampu : Bana Handaga, Dr. Ir, M.T.**

**Disusun Oleh :**

**Nama : Dwi Arianto Adi Nugroho**

**NIM : L200150124**

**Jurusan : Teknik Informatika**

**PROGRAM STUDI TEKNIK INFORMATIKA**

FAKULTAS KOMUNIKASI DAN INFORMATIKA

UNIVERSITAS MUHAMMADIYAH SURAKARTA

2017

**Instruction Set x86 Family**

1. **Pengertian Instruction Set**

Set Instruksi (Instruction Set atau Instruction Set Architecture (ISA)) didefinisikan sebagai suatu aspek dalam arsitektur komputer yang dapat dilihat oleh para pemrogram. Secara umum, ISA ini mencakup jenis data yang didukung, jenis instruksi yang dipakai, jenis register, mode pengalamatan, arsitektur memori, penanganan interupsi, eksepsi, dan operasi I/O eksternalnya (jika ada). ISA merupakan sebuah spesifikasi dari kumpulan semua kode-kode biner (opcode) yang diimplementasikan dalam bentuk aslinya (native form) dalam sebuah desain prosesor tertentu. Kumpulan opcode tersebut, umumnya disebut sebagai bahasa mesin (machine language) untuk ISA yang bersangkutan. ISA yang populer digunakan adalah set instruksi untuk chip Intel x86, IA-64, IBM PowerPC, Motorola 68000, Sun SPARC, DEC Alpha,dan lain-lain. ISA kadang-kadang digunakan untuk membedakan kumpulan karakteristik yang disebut di atas dengan mikroarsitektur prosesor, yang merupakan kumpulan teknik desain prosesor untuk mengimplementasikan set instruksi (mencakup microcode, pipeline, sistem cache, manajemen daya, dan lainnya).

Pada beberapa mesin, semua instruksi memiliki panjang yang sama, pada mesin-mesin yang lain mungkin terdapat banyak panjang berbeda. Instruksi-instruksi mungkin lebih pendek dari, memiliki panjang yang sama seperti, atau lebih panjang dari panjang word. Membuat semua instruksi memiliki panjang yang sama lebih muda dilakukan dan membuat pengkodean lebih mudah tetapi sering memboroskan ruang, karena semua instruksi dengan demikian harus sama panjang seperti instruksi yang paling panjang.

1. **Jenis-Jenis Instruksi**
2. Data Processing (Pengolahan Data) : Instruksi-instruksi aritmetika dan logika. Instruksi aritmetika memiliki kemampuan untuk mengolah data numeric, sedangkan instruksi logika beroperasi pada bit-bit word sebagai bit bukan sebagai bilangan. Operasi-operasi tersebut dilakukan terutama untuk data di register CPU.
3. Data Storage (Penyimpanan Data): Instruksi-instruksi memori. Instruksi-instruksi memori diperlukan untuk memindah data yang terdapat di memori dan register.
4. Data Movement (Perpindahan Data): Instruksi I/O. Instruksi-instruksi I/O diperlukan untuk memindahkan program dan data ke dalam memori dan mengembalikan hasil komputansi kepada pengguna.
5. Control (Kontrol): instruksi pemeriksaan dan percabangan. Instruksi-instruksi control digunakan untuk memeriksa nilai data, status komputansi dan mencabangkan ke set instruksi lain
6. **Set Instruksi Intel**
7. Set Instruksi pada Mikroprosessor 8085

Instruksi Kontrol

Berikut adalah tabel yang menunjukkan daftar instruksi Control dengan maknanya :

|  |  |  |  |
| --- | --- | --- | --- |
| Opcode | Operand | Arti | Penjelasan |
| NOP | None | No operation | Tidak ada operasi dilakukan, i.n., instruksi diambil dan diterjemahkan. |
| HLT | None | Halt and enter wait state | CPU selesai menjalankan instruksi saat ini dan berhenti eksekusi lebih lanjut. Interupsi atau ulang diperlukan untuk keluar dari negara berhenti. |
| DI | None | Disable interrupts | Interrupt mengaktifkan flip-flop reset dan semua interupsi dinonaktifkan kecuali TRAP. |
| EI | None | Enable interrupts | Interrupt mengaktifkan flip-flop diatur dan semua interupsi diaktifkan. |
| RIM | None | Read interrupt mask | instruksi ini digunakan untuk membaca status interupsi 7.5, 6.5, 5.5 dan membaca seri bit input data. |
| SIM | None | Set interrupt mask | Instruksi ini digunakan untuk melaksanakan interupsi 7.5, 6.5, 5.5, dan output data serial. |

Instruksi Logis

Tabel berikut menunjukkan daftar instruksi logis dengan maknanya.

|  |  |  |  |
| --- | --- | --- | --- |
| Opcode | Operand | Arti | Penjelasan |
| CMP | R  M | Bandingkan register atau memori dengan akumulator | Isi dari operan (register atau memori) yang M dibandingkan dengan isi akumulator. |
| CPI | 8-bit data | Bandingkan langsung dengan akumulator | Data byte kedua dibandingkan dengan isi akumulator. |
| ANA | R  M | Logical AND mendaftar atau memori dengan akumulator | Isi akumulator adalah logika AND dengan M isi register atau memori, dan hasilnya ditempatkan di akumulator. |
| ANI | 8-bit data | Logical AND langsung dengan akumulator | Isi akumulator adalah logika AND dengan data 8-bit dan hasilnya ditempatkan di akumulator. |
| XRA | R  M | Exclusive OR mendaftar atau memori dengan akumulator | Isi akumulator yang Eksklusif OR dengan M isi register atau memori, dan hasilnya ditempatkan di akumulator. |
| XRI | 8-bit data | Exclusive OR langsung dengan akumulator | Isi akumulator yang Eksklusif OR dengan data 8-bit dan hasilnya ditempatkan di akumulator. |
| ORA | R  M | Logical OR register atau memori dengan akumulator | Isi akumulator secara logis OR dengan M isi register atau memori, dan hasilnya ditempatkan di akumulator. |
| ORI | 8-bit data | Logical OR langsung dengan akumulator | Isi akumulator secara logis OR dengan data 8-bit dan hasilnya ditempatkan di akumulator. |
| RLC | None | Putar akumulator kiri | Setiap bit akumulator diputar ditinggalkan oleh satu posisi. Bit D7 ditempatkan pada posisi D0 serta bendera Carry. CY dimodifikasi menurut bit D7. |
| RRC | None | Putar akumulator kanan | Setiap bit akumulator diputar tepat oleh satu posisi. Bit D0 ditempatkan pada posisi D7 serta bendera Carry. CY dimodifikasi menurut bit D0. |
| RAL | None | Memutar akumulator kiri melalui carry | Setiap bit akumulator diputar ditinggalkan oleh satu posisi melalui bendera Carry. Bit D7 ditempatkan di bendera Carry, dan bendera Carry ditempatkan di posisi D0 paling signifikan. CY dimodifikasi menurut bit D7. |
| RAR | None | Putar akumulator kanan melalui carry | Setiap bit akumulator diputar tepat oleh satu posisi melalui bendera Carry. Bit D0 ditempatkan di bendera Carry, dan bendera Carry ditempatkan pada posisi D7 paling signifikan. CY dimodifikasi menurut bit D0. |
| CMA | None | akumulator pelengkap | Isi akumulator dilengkapi. Tidak ada flag yang terpengaruh. |
| CMC | None | carry pelengkap | Bendera Carry ini dilengkapi. Tidak ada flag lain yang terpengaruh. |
| STC | None | Set Carry | Set Carry |

Instruksi Percabangan

Tabel berikut menunjukkan daftar instruksi Percabangan  dengan maknanya.

|  |  |  |  |
| --- | --- | --- | --- |
| Opcode | Operand | Arti | Penjelasan |
| JMP | 16-bit address | Langsung tanpa syarat | Urutan program ditransfer ke alamat memori yang diberikan dalam operan. |
| |  |  |  | | --- | --- | --- | | Opcode | Description | Flag Status | | JC | Jump on Carry | CY=1 | | JNC | Jump on no Carry | CY=0 | | JP | Jump on positive | S=0 | | JM | Jump on minus | S=1 | | JZ | Jump on zero | Z=1 | | JNZ | Jump on no zero | Z=0 | | JPE | Jump on parity even | P=1 | | JPO | Jump on parity odd | P=0 | | 16-bit address | Langsung tanpa syarat | Urutan program ditransfer ke alamat memori yang diberikan dalam operan berdasarkan bendera tertentu PSW tersebut. |
| |  |  |  | | --- | --- | --- | | Opcode | Description | Flag Status | | CC | Call on Carry | CY=1 | | CNC | Call on no Carry | CY=0 | | CP | Call on positive | S=0 | | CM | Call on minus | S=1 | | CZ | Call on zero | Z=1 | | CNZ | Call on no zero | Z=0 | | CPE | Call on parity even | P=1 | | CPO | Call on parity odd | P=0 | | 16-bit address | Unconditional subroutine call | Urutan program ditransfer ke alamat memori yang diberikan dalam operan. Sebelum mentransfer, alamat dari instruksi berikutnya setelah PANGGILAN didorong ke stack. |
| RET | None | Kembali dari subrutin tanpa syarat | Urutan program ditransfer dari subrutin ke program panggilan. |
| |  |  |  | | --- | --- | --- | | Opcode | Description | Flag Status | | RC | Return on Carry | CY=1 | | RNC | Return on no Carry | CY=0 | | RP | Return on positive | S=0 | | RM | Return on minus | S=1 | | RZ | Return on zero | Z=1 | | RNZ | Return on no zero | Z=0 | | RPE | Return on parity even | P=1 | | RPO | Return on parity odd | P=0 | | None | Kembali dari subrutin bersyarat | Urutan program ditransfer dari subrutin ke program panggilan berdasarkan bendera tertentu dari PSW dan eksekusi program dimulai pada alamat baru. |
| PCHL | None | Memuat program counter dengan isi HL | Isi register H & L akan disalin ke dalam program counter. Isi H ditempatkan sebagai high-order byte dan isi L sebagai urutan byte rendah. |
| RST | 0-7 | Restart | Instruksi RST digunakan sebagai petunjuk perangkat lunak dalam program untuk mentransfer eksekusi program untuk salah satu dari delapan lokasi berikut.   |  |  | | --- | --- | | Instruction | Restart Address | | RST 0 | 0000H | | RST 1 | 0008H | | RST 2 | 0010H | | RST 3 | 0018H | | RST 4 | 0020H | | RST 5 | 0028H | | RST 6 | 0030H | | RST 7 | 0038H |   8085 memiliki tambahan 4 interupsi, yang dapat menghasilkan petunjuk RST internal dan tidak memerlukan hardware eksternal. Berikut ini adalah instruksi tersebut dan alamat Restart mereka -   |  |  | | --- | --- | | Interrupt | Restart Address | | TRAP | 0024H | | RST 5.5 | 002CH | | RST 6.5 | 0034H | | RST 7.5 | 003CH | |

Instruksi Aritmatika

Berikut adalah tabel yang menunjukkan daftar instruksi aritmatika dengan maknanya.

|  |  |  |  |
| --- | --- | --- | --- |
| Opcode | Operand | Arti | Penjelasan |
| ADD | R  M | Tambahkan register atau memori, untuk akumulator | Isi dari register atau memori ditambahkan ke isi akumulator dan hasilnya disimpan dalam akumulator.  Contoh - ADD K. |
| ADC | R  M | Tambahkan Register ke akumulator dengan carry | Isi dari register atau memori & M bendera Carry ditambahkan ke isi akumulator dan hasilnya disimpan dalam akumulator.  Contoh - ADC K |
| ADI | 8-bit data | Tambahkan langsung ke akumulator | 8-bit data ditambahkan ke isi akumulator dan hasilnya disimpan dalam akumulator.  Contoh - ADI 55K |
| ACI | 8-bit data | Tambahkan langsung ke akumulator dengan carry | 8-bit data dan bendera Carry ditambahkan ke isi akumulator dan hasilnya disimpan dalam akumulator.  Contoh - ACI 55K |
| LXI | Reg. pair, 16bit data | Memuat pasangan Register langsung | instruksi menyimpan data 16-bit ke dalam sepasang Register yang ditunjuk di operan.  Contoh - LXI K, 3025M |
| DAD | Reg. pair | Tambahkan pasangan mendaftar untuk H dan L register | Data 16-bit dari pasangan register tertentu ditambahkan ke isi register HL.  Contoh - DAD K |
| SUB | R  M | Kurangi mendaftar atau memori dari akumulator | Isi dari register atau memori yang dikurangi dari isi akumulator, dan hasilnya disimpan dalam akumulator.  Contoh - SUB K |
| SBB | R  M | Kurangi sumber dan meminjam dari akumulator | Isi dari register atau memori yang dikurangi dari isi akumulator, dan hasilnya disimpan dalam akumulator.  Contoh - SUB K |
| SUI | 8-bit data | Kurangi langsung dari akumulator | 8-bit data dikurangi dari isi akumulator & hasilnya disimpan dalam akumulator.  Contoh - SUI 55K |
| SBI | 8-bit data | Kurangi langsung dari akumulator dengan meminjam | Isi register H dipertukarkan dengan isi register D, dan isi dari register L dipertukarkan dengan isi register E.  Contoh - XCHG |
| INR | R  M | Kenaikan register atau memori dengan 1 | Isi dari pasangan daftar yang ditunjuk bertambah 1 dan hasilnya mereka disimpan di tempat yang sama.  Contoh - INR K |
| INX | R | Kenaikan Register pasangan dengan 1 | Isi dari pasangan daftar yang ditunjuk bertambah 1 dan hasilnya mereka disimpan di tempat yang sama.  contoh - INX K |
| DCR | R  M | Pengurangan register atau memori dengan 1 | Isi dari register yang ditunjuk atau memori yang dikurangi oleh 1 dan hasilnya mereka disimpan di tempat yang sama.  contoh - DCR K |
| DCX | R | Pengurangan sepasang register dengan 1 | Isi dari pasangan daftar yang ditunjuk dikurangi oleh 1 dan hasilnya mereka disimpan di tempat yang sama.  contoh -DCX K |
| DAA | None | Desimal menyesuaikan akumulator | Isi akumulator yang berubah dari nilai biner untuk dua digit 4-bit BCD.  Jika nilai low-order 4-bit dalam akumulator lebih besar dari 9 atau jika flag AC diatur, instruksi menambahkan 6 ke rendah-order empat bit.  Jika nilai-order tinggi 4-bit dalam akumulator lebih besar dari 9 atau jika flag Carry ditetapkan, instruksi menambahkan 6 ke tinggi-order empat bit.  contoh -DAA |

Instruksi Data Transfer

Following is the table showing the list of Data-transfer instructions with their meanings.

|  |  |  |  |
| --- | --- | --- | --- |
| Opcode | Operand | Arti | Penjelasan |
| MOV | Rd, Sc  M, Sc  Dt, M | Menyalin dari sumber (Sc) ke tujuan (Dt) | Instruksi ini salinan isi dari register sumber ke register tujuan tanpa perubahan apapun.  contoh -MOV K, L |
| MVI | Rd, data  M, data | Pindahkan segera 8-bit | 8-bit data disimpan dalam register tujuan atau memori.  contoh - MVI K, 55L |
| LDA | 16-bit address | Mengisi akumulator | Isi lokasi memori, yang ditentukan oleh alamat 16-bit di operan, akan disalin ke akumulator.  Contoh − LDA 2034K |
| LDAX | B/D Reg. pair | Memuat akumulator tidak langsung | Isi dari titik pasangan register yang ditunjuk ke lokasi memori. Instruksi ini salinan isi dari lokasi memori ke akumulator.  Contoh − LDAX K |
| LXI | Reg. pair, 16-bit data | Memuat pasangan register langsung | instruksi load data 16-bit dalam pasangan mendaftar ditunjuk dalam register atau memori.  Contoh − LXI K, 3225L |
| LHLD | 16-bit address | Beban H dan L register langsung | Instruksi salinan isi dari lokasi memori ditunjukkan oleh alamat ke register L dan salinan isi dari lokasi memori berikutnya dalam daftar H.  Contoh − LHLD 3225K |
| STA | 16-bit address | 16-bit address | Isi akumulator disalin ke lokasi memori yang ditentukan oleh operan.  Ini adalah instruksi 3-byte, byte kedua menetapkan alamat low-order dan byte ketiga menentukan alamat high-order.  Contoh − STA 325K |
| STAX | 16-bit address | Menyimpan akumulator tidak langsung | Isi akumulator disalin ke lokasi memori yang ditentukan oleh isi dari operan.  Contoh − STAX K |
| SHLD | 16-bit address | Toko H dan L register langsung | Isi register L disimpan dalam lokasi memori yang ditentukan oleh alamat 16-bit di operan dan isi register H disimpan ke lokasi memori berikutnya dengan incrementing operan.  Ini adalah instruksi 3-byte, byte kedua menetapkan alamat low-order dan byte ketiga menentukan alamat high-order.  Contoh − SHLD 3225K |
| XCHG | None | Efek H dan L dengan D dan E | Isi register H dipertukarkan dengan isi register D, dan isi dari register L dipertukarkan dengan isi register E.  Contoh − XCHG |
| SPHL | None | Salin H dan L register ke stack pointer | instruksi load isi dari H dan L register ke stack pointer register. Isi dari register H memberikan alamat-order tinggi dan isi dari L daftar memberikan alamat low-order.  Contoh − SPHL |
| XTHL | None | Efek H dan L dengan atas tumpukan | Isi dari L mendaftar dipertukarkan dengan lokasi tumpukan ditunjukkan oleh isi register stack pointer.  Isi dari register H ditukar dengan lokasi tumpukan berikutnya (SP + 1).  Contoh − XTHL |
| PUSH | Reg. pair | Dorong pasangan register ke stack | Isi dari pasangan mendaftar ditunjuk di operan akan disalin ke dalam stack dalam urutan berikut.  Stack pointer register dikurangi dan isi dari urutan tinggi daftar (B, D, H, A) akan disalin ke lokasi tersebut.  Stack pointer register dikurangi lagi dan isi dari register low-order (C, E, L, bendera) akan disalin ke lokasi tersebut.  Contoh − PUSH K |
| POP | Reg. pair | Pop dari stack ke pasangan register | Isi lokasi memori yang ditunjuk oleh stack pointer register disalin ke register low-order (C, E, L, status bendera) dari operan.  Stack pointer bertambah dengan 1 dan isi dari lokasi memori disalin ke high-order register (B, D, H, A) dari operan.  Stack pointer register lagi bertambah 1.  Contoh− POPK |
| OUT | 8-bit port address | Output data dari akumulator ke port dengan alamat 8 bit | Isi akumulator disalin ke dalam port I / O yang ditentukan oleh operan.  Contoh − OUT K9L |
| IN | 8-bit port address | Input data ke akumulator dari port dengan alamat 8-bit | Isi port input yang ditunjuk di operan dibaca dan dimuat ke akumulator.  Contoh − IN5KL |

<http://www.masagunglearning.com/2016/09/set-instruksi-pada-mikroprosesor-8085.html>

1. Set Instruksi pada Mikroprosessor 8086 dan 8088

Berikut ini penjelasan tentang instruksi yang digunakan pada bahasa rakitan 8086 dan 8088 atau sejenisnya :

1. **AAA** (ASCII adjust for edition) Pengaturan ASCII bagi penambahan

2. **AAD** (ASCII adjust for division) Pengaturan ASCII bagi pembagian

3. **AAM** (ASCII adjust for multipy) Pengaturan ASCII bagi perkalian

4. **AAS** (ASCII adjust for substraction) Pengaturan ASCII bagi pengurangan

5. **ADC** (add with carry) T ambahkan dengan carry

6. **ADD** (addition) Penambahan

7. **AND** (logic AND) Logik AND

8. **CALL** (CALL subroutine) Subrutin panggil

9. **CBW** (Convert byte to word) Konversikan byte ke kata

10. **CLC** (Clear Carry) Kosongkan corry flag

11. **CLD** (Clear directon flag) Kosongkan flag arah

12. **CLI** (Clear interrupt enable) Kosongkan flag penggerak interupsi

13. **DAS** (decimal adjust for substraction) Pengaturan decimal bagi pengurangan

14. **DEC** (decrement) Penurunan operand tujuan dengan 1

15. **DIV** (divide) Pembagian tak bertanda

16. **ESC** (escape) Escape digunakan sehubungan dengan suatu co-procesor external

17. **HLT** (halt) HLT menghentikan prosesor sampai saluran reset diaktifkan

18. **IDIV** (Integer division) Pembagian kilat

19. **IMUL** (Integer multiply) Perkalian kilat

20. **IN** (Input) IN (masukan) mentransfer data dari port yang disperifikasikan ke dalam register AC atau AX

21. **NC** (Increment) INC menaikkan operand tujuan dalam 1

22. **INT** (Interupt) INT (Interupsi) mengawali suatu prosedur interupsi dengan jenis yang dispesifikasikan oleh instruksi yang bersangkutan

23. **INTO** (Interupt on overflow) Instrupsi bila ada overflow INTO digunakan untuk membangkitkan suatu interupsi perangkat lunak yang bergantumg pada status flag OF

24. **IRET** (Interupt return) IRET digunakan untuk kembali dari suatu interupsi dan mendapatkan kembali IP,CS dan flag-flag dari stack

25. **JA** (Jump on above) Digunakan jika operand pertama lebih besar daripada operand kedua

26. **JNBE** (Jump) Digunakan jika pada saat pembandingan operand 1, tidak lebih kecil atausama dengan operand 2

27. **JAE** (Jump on above or equal) Digunakan pada saat pembandingan operand 1 lebih besar atau sama dengan operand 2

28. **JNB** (Jump not below) Digunakan pada saat pembandingan operand 1 tidak lebih kecil dari operand 2

29. **JB** (Jump on below) Digunakan pada saat operand 1 lebih kecil dari operand 2

30. **JNAE** (Jump on not above or equal) Digunakan pada saat operand 1 tidak lebih besar atau sama dengan operand 2

31. **JBE** (Jump on below or equal) Digunakan pada saat operand 1 lebih kecil atau sama dengan operand 2

32. **JNA** (Jump on not above) Digunakan pada saat operand 1 tidak lebih besar dari operand 2

33. **JC** (Jump on carry) Digunakan pada saat akan diprogram CF=1

34. **JCXZ** (Jump if CX=0) Digunakan bila isi register CX=0

35. **JE** (Jump on equal) Digunakan pada saat pembandingan kedua operand sama

36. **JZ** (Jump on zero) Digunakan pada saat akan diproses ZF=1

37. **JG** (Jump on greather than) Digunakan pada saat pembandingan operand 1 menunjukkan lebih besar dari operand 2

38. **JNLE** (Jump on greather or equal) Digunakan pada saat operand 1tidak lebih kecil atau sama dengan operand 2

39. **JGE** (Jump on greather or equal) Digunakan pada saat operand 1 lebih besar atau sama dengan operand 2

40. **JNL** (Jump on less) Digunakan pada saat operand 1 tidak lebih kecil dari operand 2

41. **JL** (Jump on less) Digunakan pada saat operand 1 lebih kecil dari operand 2

42. **JNGE** (Jump on not greather or equal) Digunakan pada saat operand 1 tidak lebih besar atau sama dengan operand 2

43. **JLE** (Jump on less or equal) Digunakan pada saat operand 1 lebih kecil atau sama dengan operand 2

44. **JNG** (Jump on not greather than) Digunakan pada saat operand 1 tidak lebih besar daripada operand 2

45. **JMP** (Unconditional jump) Lompatan tidak bersyarat

46. **JNC** (Jump on not carry) Digunakan pada saat diproses CF=O

47. **JNE** (Jump on not equal) Digunakan pada saat operand 1 tidak sama dari operand 2

48. **JNO** (Jump on not zero) Digunakan pada saat diproses ZF=0

49. **JNO** (Jump on not overflow) Digunakan bila tidak ada overflow (OF=O)

50. **JNS** (Jump on not sign) Digunakan pada saat diproses SF=O

51. **ZNP** (Jump on not pority) Digunakan bila tidak ada pointer

52. **JPO** (Jump on poity odd) Digunakan bila pointer ganjil

53. **O** (Jump on overflow) Digunakan pada saat ada overflow

54. **JP** (Jump on Pority equal) Digunakan bila pointer genap

55. **JS** (Jump on sign) Digunakan pada saat diproses ZF=1

56. **LAHF** (Load AH from flag) LAHF me-load bit 7,6,4,2 & 0 pada register AH masing-masing dengan isi flag SF.ZF,AF & CF

57. **LDS** (Load pointer using DS) Muatkan peminjak dengan menggunakan DS

58. **LEA** (Load effective address) LEA mentransfer opsan sumber 16 bit dalam memori ke tujuan 16 bit

59. **LES** (Load pointer using ES) Muatkan pointer dengan menggunakan ES

60. **LOCK** (Lock bas) Digunakan dalam penerapan pemakaian sumber bersama, untuk memastikan bahwa memori tidak diakses secara serentak oleh lebih dari satu proses

61. **LODS** (Load string) Muatkan string byte atau kata

62. **LOOP** Loop jika EX bukan 0

63. **LOOPE** dan LOOPZ -Loop bila sama

64. **Loop** bila 0

65. **LOOPNE/LOOPNZ** -Loop bila tidak sama (ZF=0)

66. **Loop** bila tidak nol (ZF=O)

67. **MOVE** Pindahkan byte atau kata memindahkan data 8 bit atau 16 bit

68. **MOUS** (Move string) Pindahkan data string 8 bit atau 16 bit

69. **MUL** (Multifly) Digunakan untuk mengalihkan isi bertandapada akumulator dengan suatu operand sumber yang dispesifikasikan

70. **NEG** (Negate) Mengurangkan suatu operand tujuan dari 0 & menyimpan hasil komplemen keduanya dalam tujuan

71. **NOP** (Logic NOT) NOP tidak membe rikan dampak tertentu (tidak ada operasi)

72. **NOT** (Logic OR) Operasi logic OR

73. **OR** (Logic OR) Operasi logic OR

74. **OUT** (Output) Digunakan untuk mentransfer data dari AL atau AX ke point (pangkalan)

75. **POP** (POP from stack) Keluaran data dari stack dari AL atau AX ke point (pangkalan)

76. **POPF** (POP flag) Keluarkan ke flag dan stack

77. **POSH** (Push to stack) Dorong sumber ke stack

78. **PUSHF** (Push flag) Dorong flag ke stack

79. **RCL** (Rotate lift with carry) Putar ke kirin dengan carry 1

80. **RCR** (Rotate right with carry) Putar ke kanan dengan carry 1

81. **REP** Ulangi

82. **REE** Ulangi jika sama

83. **REPZ** Ulangi jika nol

84. **REPNE** Ulangi jika tidak sama

85. **REPNZ** Ulangi jika tidak nol

86. **RET** (Return from subroutine) Digunakan untuk kembali ke subrutin

87. **ROL** (Rotate left) Rotasi (putar) ke kiri 1

88. **ROR** (Rotate right) Rotasi (putar) ke kanan 1

89. **SAHF** (store AH in flags register) Simpan AH dalam reegister flag

90. **SAL/SHL** (Shift left arithmetic/logical) Geser aritmatik/logika ke kiri 1

91. **SAR/SHR** (Shift right arithmetic/logical) Geser aritmatik/logika ke kanan 1

92. **SBB** (Substract with borrow) Kurangkan dengan borrow

93. **SCAS** (Scan string) Digunakan untuk mengurangkan string tujuan yang dialamatkan a/n regiser DI,dari AX atau AL

94. **STC** (Set carry flag) Menset carry flag (CF)

95. **STD** (Set direction flag) Menset flag arah (DF)

96. **STI** (Set interrupt enable flag) Menset flag penggerak interupsi (IF)

97. **STOS** (Substract) Memindahkan operand sumber yang terkandung dalam AX atau AL ke suatu tujuan yang dialamatkan oleh DI

98. **SUB** (Substract) Mengurangkan sumber dari operand tujuan dan hasilnya ditepatkan dalam tujuan

99. **TEST** (Logical comporison) Menguji operand-operand atau logic perbandingan

100. **WAIT** Menunggu sampai saluran test aktif

101. **XCHG** (Exchange) Pertukaran dan sumber

102. **XLAT** (Translate) Digunakan untuk menerjemahkan se-byte dalam register AL ke dalam suatu byte yang diambil dari tabel terjemahan

103. **XOR** (Logic exclusive OR) Operasi exclusive OR

<http://robisapoetra.blogspot.co.id/2013/11/bab-i-pendahuluan-a.html>

1. Set Instruksi pada ARM

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Meaning** | **Earliest CPU / Comments** |
| [ABS](https://www.heyrick.co.uk/assembler/fpops.html#unop) | Absolute Value | Floating Point 1 |
| [ACS](https://www.heyrick.co.uk/assembler/fpops.html#unop) | Arc Cosine | Floating Point 5 |
| [ADC](https://www.heyrick.co.uk/assembler/mov.html#adc) | Add with Carry | - |
| ADC | Thumb: Add with Carry | Thumb |
| [ADD](https://www.heyrick.co.uk/assembler/mov.html#add) | Add | - |
| ADD | Thumb: Add | Thumb |
| [ADF](https://www.heyrick.co.uk/assembler/fpops.html#binop) | Add | Floating Point 1 3 |
| [ADR](https://www.heyrick.co.uk/assembler/pseudo.html#adr) | Get address of object (within 4K) | This is an assembler pseudo-instruction |
| [ADRL](https://www.heyrick.co.uk/assembler/pseudo.html#adrl) | Get address of object (beyond 4K) | This is an assembler pseudo-instruction |
| [ALIGN](https://www.heyrick.co.uk/assembler/pseudo.html#align) | Set the program counter to the next word boundary | This is an assembler pseudo-instruction |
| [AND](https://www.heyrick.co.uk/assembler/mov.html#and) | Logical AND | - |
| AND | Thumb: Logical AND | Thumb |
| [ASL](https://www.heyrick.co.uk/assembler/shift.html#asl) | Arithmetic Shift Left | This is an option, not an instruction |
| [ASN](https://www.heyrick.co.uk/assembler/fpops.html#unop) | Arc Sine | Floating Point 5 |
| [ASR](https://www.heyrick.co.uk/assembler/shift.html#asr) | Arithmetic Shift Right | This is an option, not an instruction;  available on Thumb. |
| [ATN](https://www.heyrick.co.uk/assembler/fpops.html#unop) | Arc Tangent | Floating Point 5 |
| [B](https://www.heyrick.co.uk/assembler/bl.html#b) | Branch | - |
| B | Thumb: Branch | Thumb |
| [BIC](https://www.heyrick.co.uk/assembler/mov.html#bic) | Bit Clear | - |
| BIC | Thumb: Bit Clear | Thumb |
| BKPT | Thumb: Breakpoint | Thumb |
| [BL](https://www.heyrick.co.uk/assembler/bl.html#bl) | Branch with Link | - |
| BL | Thumb: Long Branch with Link | Thumb |
| BLX | Thumb: Branch with Link and Exchange | Thumb |
| BX | Thumb: Branch and Exchange | Thumb |
| [CDP](https://www.heyrick.co.uk/assembler/coprocmnd.html#cdp) | Co-processor data operation | - |
| CDP2 | CDP, non-conditional so more co-processor commands possible | ARMv5 |
| CLZ | Count Leading Zeros | ARMv5 |
| [CMF](https://www.heyrick.co.uk/assembler/fpops.html#cmf) | Compare floating point value | Floating Point 1 3 |
| [CMN](https://www.heyrick.co.uk/assembler/cmp.html#cmn) | Compare negated values | - |
| CMN | Thumb: Compare negated values | Thumb |
| [CMP](https://www.heyrick.co.uk/assembler/cmp.html#cmp) | Compare values | - |
| CMP | Thumb: Compare values | Thumb |
| [CNF](https://www.heyrick.co.uk/assembler/fpops.html#cnf) | Compare negated floating point values | Floating Point 1 |
| [COS](https://www.heyrick.co.uk/assembler/fpops.html#unop) | Cosine | Floating Point 5 |
| [DCx](https://www.heyrick.co.uk/assembler/pseudo.html#dcx) | Define byte (B), halfword (W), word (D), string (S), or floating point (F) value. Some assemblers allow DCFS, DCFD, etc for FP precision. | This is an assembler pseudo-instruction |
| [DVF](https://www.heyrick.co.uk/assembler/fpops.html#binop) | Divide | Floating Point 1 3 |
| [EOR](https://www.heyrick.co.uk/assembler/mov.html#eor) | Exclusive-OR two values | - |
| EOR | Thumb: Logical Exclusive-OR | Thumb |
| [EQUx](https://www.heyrick.co.uk/assembler/pseudo.html#equx) | Define byte (B), halfword (W), word (D), string (S), or floating point (F) value. Some assemblers allow EQUFS, EQUFD, etc for FP precision. | This is an assembler pseudo-instruction |
| [EXP](https://www.heyrick.co.uk/assembler/fpops.html#unop) | Exponent | Floating Point 5 |
| FABS | VFP: Absolute | VFP |
| FADD | VFP: Addition | VFP |
| FCMP | VFP: Compare | VFP |
| FCVTDS | VFP: Single to Double | VFP |
| FCVTSD | VFP: Double to Single | VFP |
| FCPY | VFP: Copy [like MVF] | VFP |
| FDIV | VFP: Division | VFP |
| [FDV](https://www.heyrick.co.uk/assembler/fpops.html#binop) | Fast Divide | Floating Point 1 |
| [FIX](https://www.heyrick.co.uk/assembler/fpops.html#fix) | Convert floating value to an integer | Floating Point 1 3 |
| FLD | VFP: Load VFP registers | VFP |
| FLDMDB | VFP: Load multiple VFP registers, decr. before | VFP |
| FLDMIA | VFP: Load multiple VFP registers, incr. after | VFP |
| [FLT](https://www.heyrick.co.uk/assembler/fpops.html#flt) | Convert integer to a floating value | Floating Point 1 3 |
| FMAC | VFP: Multiply with Accumulate | VFP |
| FMDHR | VFP: Transfer ARM register to upper half of Double | VFP |
| FMDLR | VFP: Transfer ARM register to lower half of Double | VFP |
| FMRDH | VFP: Transfer upper half of Double to ARM register | VFP |
| FMRDL | VFP: Transfer lower half of Double to ARM register | VFP |
| [FML](https://www.heyrick.co.uk/assembler/fpops.html#binop) | Fast multiply | Floating Point 1 |
| FMSC | VFP: Multiply with Negate and Accumulate | VFP |
| FMRS | VFP: Transfer Single to ARM register | VFP |
| FMSR | VFP: Transfer ARM register to Single | VFP |
| FMUL | VFP: Multiply | VFP |
| FMRX | VFP: Transfer VFP system register to ARM register | VFP |
| FMSTAT | VFP: Transfer FPSCR flags to CPSR | VFP |
| FMXR | VFP: Transfer ARM register to VFP system register | VFP |
| FNEG | VFP: Copy Negative [like MVN] | VFP |
| FNMAC | VFP: Multiply with Deduct | VFP |
| FNMSC | VFP: Multiply with Negate and Deduct | VFP |
| FNMUL | VFP: Negative Multiply | VFP |
| [FRD](https://www.heyrick.co.uk/assembler/fpops.html#binop) | Fast reverse divide | Floating Point 1 |
| FSITO | VFP: Signed Integer to Float | VFP |
| FSQRT | VFP: Square Root | VFP |
| FST | VFP: Save VFP registers | VFP |
| FSTMDB | VFP: Save multiple VFP registers, decr. before | VFP |
| FSTMIA | VFP: Save multiple VFP registers, incr. after | VFP |
| FSUB | VFP: Subtraction | VFP |
| FTOSI | VFP: Float to Signed Integer | VFP |
| FTOUI | VFP: Float to Unsigned Integer | VFP |
| FUITO | VFP: Unsigned Integer to Float | VFP |
| [LDC](https://www.heyrick.co.uk/assembler/coprocmnd.html#ldc) | Load from memory to co-processor | - |
| LDC2 | LDC, non-conditional so more co-processor commands possible | ARMv5 |
| [LDF](https://www.heyrick.co.uk/assembler/fpops.html#ldf) | Load floating point value | Floating Point 1 3 |
| [LDM](https://www.heyrick.co.uk/assembler/str.html#ldm) | Load multiple registers | - |
| LDMIA | Thumb: Load multiple registers | Thumb |
| [LDR](https://www.heyrick.co.uk/assembler/str.html#ldr) | Load register (32 bit) | - |
| LDR | Thumb: Load register (32 bits?) | Thumb |
| [LDRB](https://www.heyrick.co.uk/assembler/str.html#ldr) | Load byte (8 bit) into register | - |
| LDRB | Thumb: Load byte (8 bit) into register | Thumb |
| LDRH | Load halfword (16 bit) into register | StrongARM |
| LDRH | Thumb: Load halfwit (boo!) into register | Thumb |
| LDRSB | Load signed byte (sign + 7 bit) into register | StrongARM |
| LDRSB | Thumb: Load signed byte (sign + 7 bit) into register | Thumb |
| LDRSH | Load signed halfword (sign + 15 bit) into register | StrongARM |
| LDRSH | Thumb: Load signed halfword (sign + 15 bit) into register | Thumb |
| [LFM](https://www.heyrick.co.uk/assembler/fpops.html#lfm) | Load multiple floating point values | Floating Point 1 |
| [LGN](https://www.heyrick.co.uk/assembler/fpops.html#unop) | Logarithm to base e | Floating Point 5 |
| [LOG](https://www.heyrick.co.uk/assembler/fpops.html#unop) | Logarithm to base 10 | Floating Point 5 |
| [LSL](https://www.heyrick.co.uk/assembler/shift.html#lsl) | Logical Shift Left | This is an option, not an instruction;  available on Thumb. |
| [LSR](https://www.heyrick.co.uk/assembler/shift.html#lsr) | Logical Shift Right | This is an option, not an instruction;  available on Thumb. |
| [MCR](https://www.heyrick.co.uk/assembler/coprocmnd.html#mcr) | Co-processor register transfer (ARM to co-processor) | - |
| MCR2 | MCR, non-conditional so more co-processor commands possible | ARMv5 |
| MCRR | MCR, with two registers transferred at one time | ARMv5TE |
| [MLA](https://www.heyrick.co.uk/assembler/mul.html#mla) | Multiply with Accumulate | - |
| [MNF](https://www.heyrick.co.uk/assembler/fpops.html#mnf) | Move negated | Floating Point 1 |
| [MOV](https://www.heyrick.co.uk/assembler/mov.html#mov) | Move value/register into a register | - |
| MOV | Thumb: Move value/register into a register | Thumb |
| [MRC](https://www.heyrick.co.uk/assembler/coprocmnd.html#mrc) | Co-processor register transfer (co-processor to ARM) | - |
| MRC2 | MRC, non-conditional so more co-processor commands possible | ARMv5 |
| MRRC | MRC, with two registers transferred at one time | ARMv5TE |
| [MRS](https://www.heyrick.co.uk/assembler/psr.html#32bit) | Move status flags to a register | ARM 6 |
| [MSR](https://www.heyrick.co.uk/assembler/psr.html#32bit) | Move contents of a register to the status flags | ARM 6 |
| [MUF](https://www.heyrick.co.uk/assembler/fpops.html) | Multiply | Floating Point 1 3 |
| [MUL](https://www.heyrick.co.uk/assembler/mul.html#binop) | Multiply | - |
| MUL | Thumb: Multiply | Thumb |
| [MVF](https://www.heyrick.co.uk/assembler/fpops.html#mvf) | Move value/float register into a float register | Floating Point 1 3 |
| [MVN](https://www.heyrick.co.uk/assembler/mov.html#mvn) | Move negated | - |
| MVN | Thumb: Move negated | Thumb |
| NEG | Thumb Negate | Thumb |
| NOP | Thumb: No Operation | Thumb |
| [NRM](https://www.heyrick.co.uk/assembler/fpops.html#unop) | Normalise | Floating Point 1 |
| [OPT](https://www.heyrick.co.uk/assembler/opt.html) | Select assembly options | This is an assembler pseudo-instruction |
| [ORR](https://www.heyrick.co.uk/assembler/mov.html#orr) | Logical OR | - |
| ORR | Thumb: Logical OR | Thumb |
| PLD | PreLoaD | ARMv5 |
| [POL](https://www.heyrick.co.uk/assembler/fpops.html#binop) | Polar Angle | Floating Point 5 |
| POP | Thumb: Pop registers from stack | Thumb |
| [POW](https://www.heyrick.co.uk/assembler/fpops.html#binop) | Power | Floating Point 5 |
| PUSH | Thumb: Push registers onto stack | Thumb |
| QADD | Add, saturating | ARMv5E |
| QDADD | Add, double saturating | ARMv5E |
| QDSUB | Subtract, double saturating | ARMv5E |
| QSUB | Subtact, saturating | ARMv5E |
| [RDF](https://www.heyrick.co.uk/assembler/fpops.html#binop) | Reverse Divide | Floating Point 1 |
| [RFC](https://www.heyrick.co.uk/assembler/fpops.html#rfc) | Read FP control register | Floating Point 1 4 |
| [RFS](https://www.heyrick.co.uk/assembler/fpops.html#rfs) | Read FP status register | Floating Point 1 3 |
| [RMF](https://www.heyrick.co.uk/assembler/fpops.html#rmf) | Remainder | Floating Point 2 3 |
| [RND](https://www.heyrick.co.uk/assembler/fpops.html#unop) | Round to integral value | Floating Point 2 3 |
| [ROR](https://www.heyrick.co.uk/assembler/shift.html#ror) | Rotate Right | This is an option, not an instruction;  available on Thumb. |
| [RPW](https://www.heyrick.co.uk/assembler/fpops.html#binop) | Reverse Power | Floating Point 5 |
| [RRX](https://www.heyrick.co.uk/assembler/shift.html#rrx) | Rotate Right with extend | This is an option, not an instruction |
| [RSB](https://www.heyrick.co.uk/assembler/mov.html#rsb) | Reverse Subtract | - |
| [RSC](https://www.heyrick.co.uk/assembler/mov.html#rsc) | Reverse Subtract with Carry | - |
| [RSF](https://www.heyrick.co.uk/assembler/fpops.html#rsf) | Reverse Subtract | Floating Point 1 |
| [SBC](https://www.heyrick.co.uk/assembler/mov.html#sbc) | Subtract with Carry | - |
| SBC | Thumb: Subtract with Carry | Thumb |
| [SFM](https://www.heyrick.co.uk/assembler/fpops.html#sfm) | Store Muliple Floating point values | Floating Point 1 |
| [SIN](https://www.heyrick.co.uk/assembler/fpops.html#unop) | Sine | Floating Point 5 |
| SMLA | Signed Multiply with Accumulate of 16 bit \* 16 bit values | ARMv5E |
| SMLAL | Signed Long (sign + 63 bit) Multiply with Accumulate | StrongARM |
| SMLAL | Signed Multiply with Accumulate of 16 bit \* 16 bit values, result is sign extended to 32 bits, then added to a 64 bit value. | ARMv5E |
| SMLAW | Signed Multiply with Accumulate of 32 bit \* 16 bit values | ARMv5E |
| SMUL | Signed Multiply of 16 bit \* 16 bit values | ARMv5E |
| SMULL | Signed Long (sign + 63 bit) Multiply | StrongARM |
| SMULW | Signed Multiply of 32 bit \* 16 bit values | ARMv5E |
| [SQT](https://www.heyrick.co.uk/assembler/fpops.html#unop) | Square Root | Floating Point 2 3 |
| [STC](https://www.heyrick.co.uk/assembler/coprocmnd.html#stc) | Co-processor data transfer | - |
| STC2 | STC, non-conditional so more co-processor commands possible | ARMv5 |
| [STF](https://www.heyrick.co.uk/assembler/fpops.html#stf) | Store floating point value | Floating Point 1 3 |
| [STM](https://www.heyrick.co.uk/assembler/str.html#stm) | Store multiple registers | - |
| STMIA | Thumb: Store multiple registers | Thumb |
| [STR](https://www.heyrick.co.uk/assembler/str.html#str) | Store a register (32 bit) | - |
| STR | Thumb: Store register (32 bit?) | Thumb |
| [STRB](https://www.heyrick.co.uk/assembler/str.html#str) | Store a byte (8 bit) from a register | - |
| STRB | Thumb: Store byte (8 bit) | Thumb |
| STRH | Store a halfword (16 bit) from a register | StrongARM |
| STRH | Thumb: Store halfword (16 bit) | Thumb |
| STRSB | Store a signed byte (sign + 7 bit) from a register | StrongARM |
| STRSH | Store a signed half-word (sign + 15 bit) from a register | StrongARM |
| [SUB](https://www.heyrick.co.uk/assembler/mov.html#sub) | Subtract | - |
| SUB | Thumb: Subtract | Thumb |
| [SUF](https://www.heyrick.co.uk/assembler/fpops.html#binop) | Subtract | Floating Point 1 3 |
| [SWI](https://www.heyrick.co.uk/assembler/swi.html#swi) | Cause a SoftWare Interrupt | - |
| SWI | Thumb: SoftWare Interrupt | Thumb |
| [SWP](https://www.heyrick.co.uk/assembler/mov.html#swp) | Swap register with memory | ARM 3 |
| [TAN](https://www.heyrick.co.uk/assembler/fpops.html#unop) | Tangent | Floating Point 5 |
| [TEQ](https://www.heyrick.co.uk/assembler/cmp.html#teq) | Test Equivalence (notional EOR) | - |
| [TST](https://www.heyrick.co.uk/assembler/cmp.html#tst) | Test bits (notional AND) | - |
| TST | Thumb: Test bits | Thumb |
| UMLAL | Unsigned Long (64 bit) Multiply with Accumulate | StrongARM |
| UMULL | Unsigned Long (64 bit) Multiply | StrongARM |
| [URD](https://www.heyrick.co.uk/assembler/fpops.html#unop) | Unnormalised round | Floating Point 1 |
| [WFC](https://www.heyrick.co.uk/assembler/fpops.html#wfc) | Write FP control register | Floating Point 1 4 |
| [WFS](https://www.heyrick.co.uk/assembler/fpops.html#wfs) | Write FP status register | Floating Point 1 3 |