Chapter 2

Instruction Set

RUNGTA COLLEGE OF ENGG AND TECH.,BHILAI
DEPT OF ELECTRONICS AND TELE. ENGG

Instruction Set Architecture

- Does not include information as to how microprocessor is designed or implemented
- Includes microprocessor instruction set, which would be the set of all assembly languages instructions.
- Also includes the complete set of accessible registers.

Levels of Programming Languages

- Programming languages are divided into three categories.
- High level languages hide the details of the computer and operating system.
- Are also referred to as platform-independent.
- Examples include C++, Java, and Fortran

Levels of Programming L

- Assembly language is an example of a lower level language.
- Each microprocessor has its own assembly language
- A program written in the assembly language of one microprocessor cannot be run on a different microprocessor

Levels of Programming Languages

- Backward compatibility used in order to have old programs that ran on a old microprocessor, can run on a newer model.
- Assembly language can manipulate the data stored in a microprocessor.
- Assembly language are not platform independent

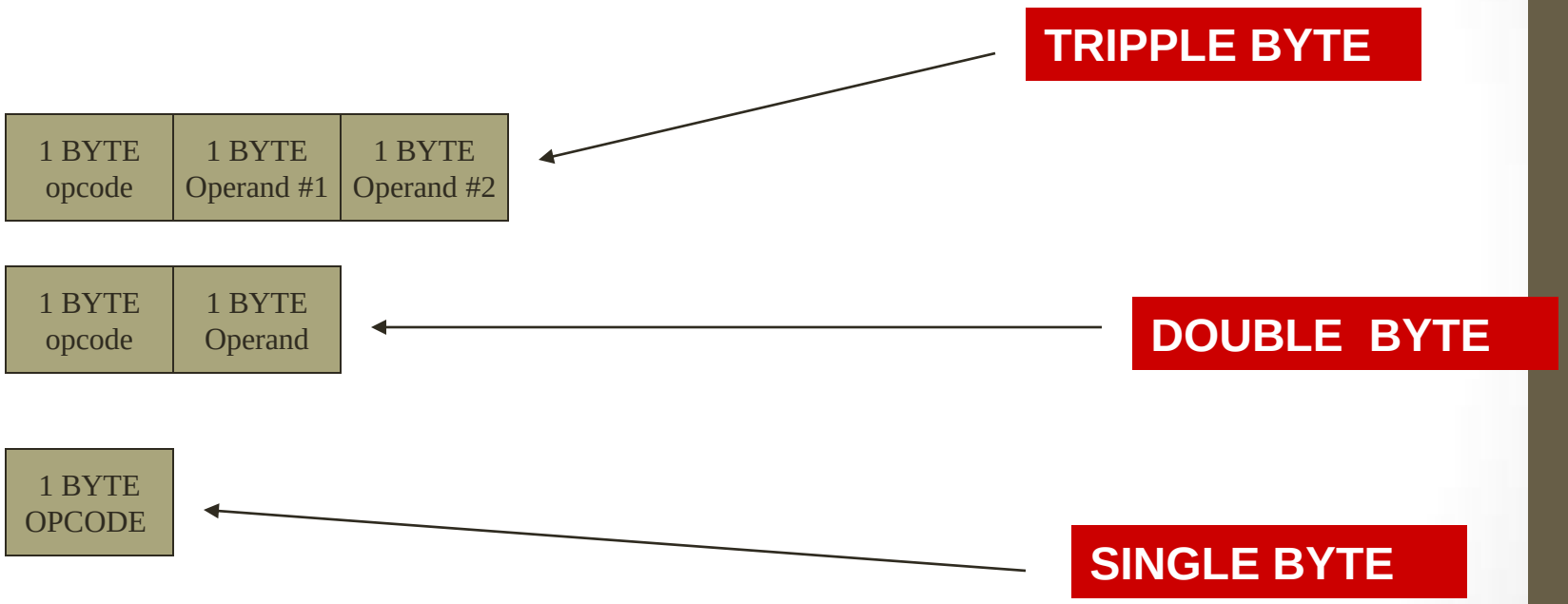
Assemblers

- Every statement in assembly language however corresponds to one unique machine code instruction.
- The assembler converts source code to object code, and then the linking, and the loading of procedures occur.

VARIOUS GROUPS OF INSTRUCTIONS

- DATA TRANSFER
- ARITHMATIC
- LOGICAL
- STACK
- BRANCHING
- MACHINE CONTROL
- INPUT-OUTPUT INSTRUCTION

3.2.4 Instruction Formats



Addressing Modes

- Microprocessor needs memory address to access data from the memory.
- Assembly language may use several addressing modes to accomplish this task.

Addressing Modes (contd)

1 Direct Mode

- Instruction includes memory access.
- CPU accesses that location in memory.

Example:

LDA 5000

Reads the data from memory location 5000, and stores the data in the CPU's accumulator.

Addressing Modes (contd)

2 **Register Indirect add Mode**

- Address specified in instruction contains address where the operand resides.

Example:

MOV M,A

Addressing Modes (contd)

3 Register Direct addressing Modes

- Does not specify a memory address. Instead specifies a register.

  Example:

  MOV B,C

Addressing Modes (contd)

4 Immediate Mode

- The operand specified in this mode is the actual data it self.

Example:

MVI A, 34H

Addressing Modes (contd)

5 Implicit Mode

- Does not exactly specify an operand. Instruction implicitly specifies the operand because it always applies to a specific register.

Example:

DAA, HLT

3.3 Instruction Set Architecture Design

To design a optimal microprocessor, the following questions and issues have to be addressed in order to come up with an optimized instruction set architecture for the CPU:

1. Completeness; does the instruction set have all of the instructions a program needs to perform its required task.
2. Issue of orthogonality, the concept of two instructions not overlapping, and thus not performing the same function.
3. The amount of registers to be added. More registers enables a CPU to run faster, since it can access and store data on registers, instead of the memory, which in turn enables a CPU to run faster. Having too many registers adds unnecessary hardware.
4. Does this processor have to be backward compatible with other microprocessors.
5. What types and sizes of data will the microprocessor deal with?
6. Are interrupts needed?
7. Are conditional instructions needed?

3.4 Creating a simple Instruction Set

Designing a simple microprocessor fit for maybe a microwave will involve integrating the following models:

1. Memory model
2. Register model
3. Instruction set

3.4.1 Memory Model

- Microprocessor can access 64 K or 2^16 byes of memory
- Each byte has 8 bits or 64K x 8 of memory.
- I/O is treated as memory access, thus requires same instruction to access I/O as it does to access memory

3.4.2 Registers

- Three registers in this microprocessor
- First register is 8-bit accumulator where the result is stored. Also provides one of the operands for instructions requiring two operands.
- Second register R is a 8-bit register that provides the second operands, and also stores in result so that the accumulator can gain access to it.
- Third register is a Z register which is 1 bit. It is either 0 or 1. If a result of a instruction is 0 then the register is set to 1 otherwise it is set to 0.

3.4.3 Instruction Set

Instruction	Instruction Code	Operation
NOP	0000 0000	No Operation
LDAC	0000 0001 Γ	AC = M[Γ]
STAC	0000 0010 Γ	M[Γ] = AC
MVAC	0000 0011	R = AC
MOVR	0000 0100	AC = R
JUMP	0000 0101 Γ	GOTO Γ
JMPZ	0000 0110 Γ	IF (Z = 1) THEN GOTO Γ
JPNZ	0000 0111 Γ	IF (Z = 0) THEN GOTO Γ
ADD	0000 1000	AC=AC+R,If (AC+R=0) Then Z=1 Else Z = 0
SUB	0000 1001	AC-AC-R,If(AC-R=0) Then Z=1 Else Z = 0
INAC	0000 1010	AC=AC+1,If(AC+1=0) Then Z=1 Else Z = 0
CLAC	0000 1011	AC = 0, Z =1
AND	0000 1100	AC=AC∧R, If(AC ∧R=0) Then Z=1 Else Z=0
OR	0000 1101	AC=AC∨R, If(AC∨R=0) Then Z=1 Else Z=0
XOR	0000 1110	AC=AC⊕R,If(AC⊕R=0) Then Z=1 Else Z=0
NOT	0000 1111	AC=AC',If(AC'=0) Then Z=1 Else Z=0

3.4.3 Instruction Set (contd)

Note: LDAC uses direct addressing mode. MOVR uses the implicit addressing mode. JUMP uses immediate addressing mode.

3.4.4 Implementation

1 + 2 + … + n, or

Total = 0

For I = 1 TO N do (Total = Total + I);

Break Down:

1: Total = 0, I = 0

2: I = I + 1

3: Total = Total + I

4: If I ≠ n THEN GOTO 2

3.4.4 Implementation (contd)

	CLAC	Clear Accumulator
	STAC *total*	Store value 0 to address total
	STAC i	Store value 0 to address i
Loop:	LDAC i	Load contents of address i into accumulator
	INAC	Add 1 to the accumulator
	STAC i	Store result from accumulator back to address i
	MVAC	Move result from accumulator into Register R
	LDAC *total*	Load Total into accumulator
	ADD	Add contents of Register R and accumulator and store it in accumulator
	STAC *total*	Store Total back to address total
	LDAC *n*	Load n into accumulator
	SUB	Subtract R (R = i) from AC (AC = n)
	JPNZ *Loop*	If result is not zero then jump back to loop:

3.4.4 Implementation (contd)

Instruction	1st Loop	2nd Loop	3rd Loop	4th Loop	5th Loop
CLAC	AC = 0				
STAC total	Total = 0				
STAC I	I = 0				
LDAC I	AC = 0	AC = 1	AC = 2	AC = 3	AC = 4
INAC	AC = 1	AC = 2	AC = 3	AC = 4	AC = 5
STAC I	I = 1	I = 2	I = 3	I = 4	I = 5
MVAC	R = 1	R = 2	R = 3	R = 4	R = 5
LDAC total	AC = 0	AC = 1	AC = 3	AC = 6	AC = 10
ADD	AC = 1	AC = 3	AC = 6	AC = 10	AC = 15
STAC total	Total = 1	Total = 3	Total = 6	Total = 10	Total = 15
LDAC n	AC = 5	AC = 5	AC = 5	AC = 5	AC = 5
SUB	AC = 4, Z = 0	AC = 3, Z = 0	AC = 2, Z = 0	AC = 1, Z = 0	AC = 0, Z = 1
JPNZ Loop	JUMP	JUMP	JUMP	JUMP	NO JUMP

3.4.5 Analysis of Instruction Set, and Implementation

- Cannot have value greater then 255, therefore n has to be less then or equal to 22
- Is it complete? For simple hardware, maybe. Not enough to be implemented in a PC.
- Fairly orthogonal; however by eliminating OR and implementing by AND and NOT, we can reduce the amount of hardware used.
- Not enough registers.

3.5 8085 Microprocessor Instruction Set Archtecture

- Processor has practical applications. Examples include the Sojourner robot.
- Contains several registers including the accumulator register, A.
- Other registers include B,C,D,E,H,L.
- Some are accessed as pairs. Pairs are not arbitrary. B and C, D and E, H and L.
- SP is a 16 bit stack pointer register pointing to the top of the stack.

3.5 8085 Microprocessor Instruction Set Archtecture

Contains five flags known as flag registers:

- Sign flag, S indicates sign of a value
- Zero flag, Z, tells if a arithmetic or logical instruction produced 0 for a result.
- Parity flag, P, is set to 1 if result contains even number of 1's
- Carry flag, CY, is set when an arithmetic operation generates a carry out.

3.5 8085 Microprocessor Instruction Set Archtecture

- Auxiliary carry flag, generates a carry out from a lower half of a result to a upper half.

  Example:

  0000 1111 + 0000 1000 = 0001 0111

- IM register used for enable and disable interrupts, and to check pending interrupts.

3.5.2 8085 Microprocessor Instruction Set

Contains a total of 74 instructions.

R, R1, R2	8 bit registers representing A, B, C, D, E, H or L
M	Indicates memory location
RP	Indicates register pair such as BC, DE, HL, SP
Γ	16 bit address representing address or data value.
n	8-bit address or data value stored in memory immediately after the opcode
Cond	Condition for conditional instructions. NZ (Z = 0), Z (Z = 1),P (S = 0), N (S = 1), PO (P = 0), PE (P = 1), NC (CY = 0), C (CY=1)

3.5.2 Data movemement instruction for the 80855 microprocessor

Instruction	Operation
NOP	No operation
MOV r1, r2	r1 = r2
MOV r, M	r1 = M[HL]
MOV M, r	M[HL] = r
MVI r, n	r = n
MVI M, n	M[HL] = n
LXI rp, Γ	rp = Γ
LDA Γ	A = M[Γ]
STA Γ	M[Γ] = A
LHLD Γ	HL = M[Γ], M[Γ + 1]
SHLD Γ	M[Γ], M[Γ + 1] = HL
LDAX rp	A = M[rp] (rp = BC, DE)
STAX rp	M[rp] = A (rp = BC, DE)
XCHG	DE ↔ HL
PUSH rp	Stack = rp (rp ≠ SP)
PUSH PSW	Stack = A, flag register
POP rp	rp = Stack (rp ≠ SP)
POP PSW	A, flag register = Stack
XTHL	HL ↔ Stack
SPHL	SP = HL
IN n	A = input port n
OUT n	Output port n = A

3.5.2 Data operation instruction for the 80855 microprocessor

Instruction	Operation	Flags
ADD r	A = A + r	All
ADD M	A = A + M[HL]	All
ADI n	A = A + n	All
ADC r	A = A + r + CY	All
ADC M	A = A + M[HL] + CY	All
ACI n	A = A + n + CY	All
SUB r	A = A - r	All
SUB M	A = A - M[HL]	All
SUI n	A = A - n	All
SBB r	A = A - r - CY	All
SBB M	A = A - M[HL] - CY	All
SBI n	A = A - n - CY	All
INR r	r = r + 1	Not CY
INR M	M[HL] = M[HL] + 1	Not CY
DCR r	r = r - 1	Not CY
DCR M	M[HL] = M[HL] - 1	Not CY
INX rp	rp = rp + 1	None
DCX rp	rp = rp - 1	None
DAD rp	HL = HL + rp	CY
DAA	Decimal adjust	All
ANA r	A = A $\wedge$ r	All
ANA M	A = A $\wedge$ M[HL]	All

3.5.2 Data operation instruction for the 80855 microprocessor

Instruction	Operation	Flags
ANI n	A = A ∧ n	All
ORA r	A = A ∨ r	All
ORA M	A = A ∨ M[HL]	All
ORI n	A = A ∨ n	All
XRA r	A = A ⊕ r	All
XRA M	A = A ⊕ M[HL]	All
XRI n	A = A ⊕ n	All
CMP r	Compare A and r	All
CMP M	Compare A and M[HL]	All
CPI n	Compare A and n	All
RLC	CY = A7, A = A(6-0), A7	CY
RRC	CY = A0, A = A0, A(7-1)	CY
RAL	CY, A = A, CY	CY
RAR	A, CY = CY, A	CY
CMA	A = A’	None
CMC	CY = CY’	CY
STC	CY = 1	CY

3.5.2 Program control instruction

Instruction	Operation
JUMP Γ	GOTO Γ
J cond Γ	If condition is true then GOTO Γ
PCHL	GOTO address HL
CALL Γ	Call subroutine at Γ
C cond Γ	If condition is true then call subroutine at Γ
RET	Return from subroutine
R cond	If condition is true then return from subroutine
RST n	Call subroutine at 8*n (n = 5.5, 6.5, 7.5)
RIM	A = IM
SIM	IM = A
DI	Disable interrupts
EI	Enable interrupts
HLT	Halt the CPU

3.5.3. A Simple 8085 Program

1: i = n, sum = 0

2: sum = sum + i, i = i - 1

3: IF i $\neq$ 0 then GOTO 2

4: total = sum

3.5.3 A Simple 8085 Program (contd)

```
        LDA n
        MOV B, A
        XRA A
Loop:   ADD B
        DCR B
        JNZ Loop
        STA total
```

$i = n$

$sum = A \oplus A = 0$

$sum = sum + i$

$i = i - 1$

IF $i \neq 0$ THEN GOTO *Loop*

$total = sum$

3.5.3 Execution trace

Instruction	1st Loop	2nd Loop	3rd Loop	4th Loop	5th Loop
LDA n MOV B, A	B = 5				
XRA A	A = 0				
ADD B	A = 5	A = 9	A = 12	A = 14	A = 15
DCR B	B = 4, Z = 0	B = 3, Z = 0	B = 2, Z = 0	B = 1, Z = 0	B = 0, Z = 1
JNZ *Loop*	JUMP	JUMP	JUMP	JUMP	NO JUMP
STA *total*					*total* = 15

3.5.4 Analyzing the 8085 ISA

- Instruction set more complete then the simple CPU, however not sufficient enough for a PC.
- Able to use subroutines, and interrupts
- It is fairly orthogonal.
- Has sufficient number of registers

8085 ASSEMBLER DIRECTIVES

Assembler directives are instructions to the assembler concerning the program being assembled. They are not translated into machine code or assigned any memory locations in the object file.

Assembler Directive	Example	Description
ORG (origin)	org 20	The next block of instructions or data should be stored in memory locations starting at 2010. Either hex or decimal numbers are acceptable.
END	end start	End of assembly. A HLT instruction may suggest the end of a program, but does not necessarily mean it is the end of assembly. "start" is the label at the beginning of the program*.

- EQU
- (equate)
- lookup equ 2 The value of the term, lookup, is equal
- to 2. lookup's value may be referred
- by name in the program. Similar to a
- constant statement.
- inbuf equ 2099 The value of the term, inbuf, is 2099.
- This may be the memory location used
- as an input buffer.
- DB
- (define byte)
- data: db 34
- or
- data: db 34
- db A2
- db 93
- Initialises an area byte by byte.
- Assembled bytes of data are stored in
- successive memory locations until all
- values are stored. The label is
- optional and may be used as the
- memory location of the beginning of
- the data.

- DW
- (define word)
- long: dw 2050 Initialises an area two bytes at a time.
- DS
- (define storage)
- table: ds 10 Reserves a specified number of
- memory locations. In this example,
- 10 memory locations are reserved for
- "table". The label may be used as the
- memory location of the beginning of
- the block of memory.