Acknowledgment

The instructor thanks the author(s) of the presentation for sharing the information

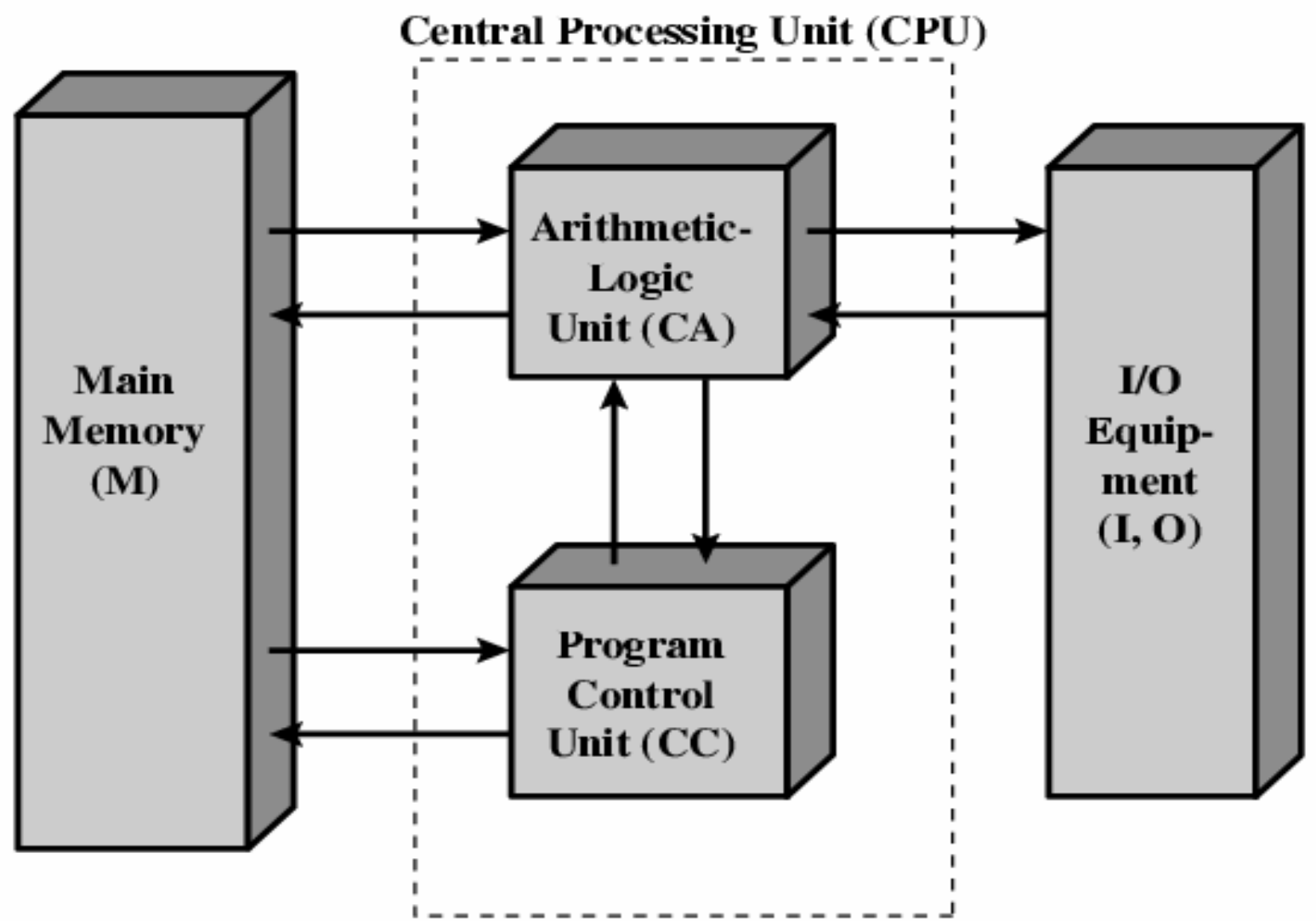
Introduction

- Computer architecture is defined by computer structure and behavior as seen by their programmer that uses machine language instructions.
- This includes the instruction formats, addressing modes, instruction set and general organization of the CPU registers

Organization of Von-Neumann Machine (IAS Computer)

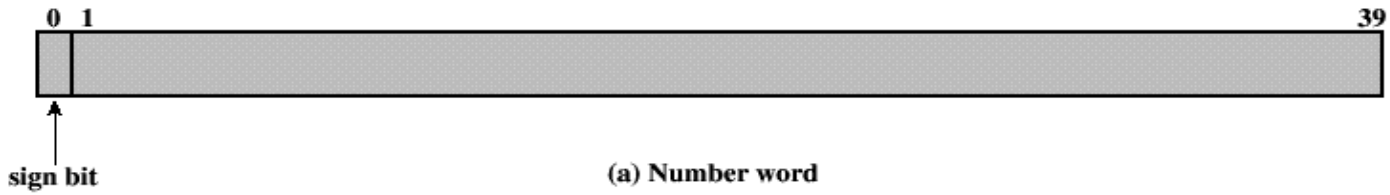
- The task of entering and altering programs for ENIAC was extremely tedious
- Stored program concept – says that the program is stored in the computer along with any relevant data
- A stored program computer consists of a p*rocessing unit* and an attached *memory system*.
- The processing unit consists of d*ata-path* and *control*. The data-path contains *registers* to hold data and *functional units*, such as arithmetic logic units and shifters, to operate on data.

Structure of Von Neumann Machine

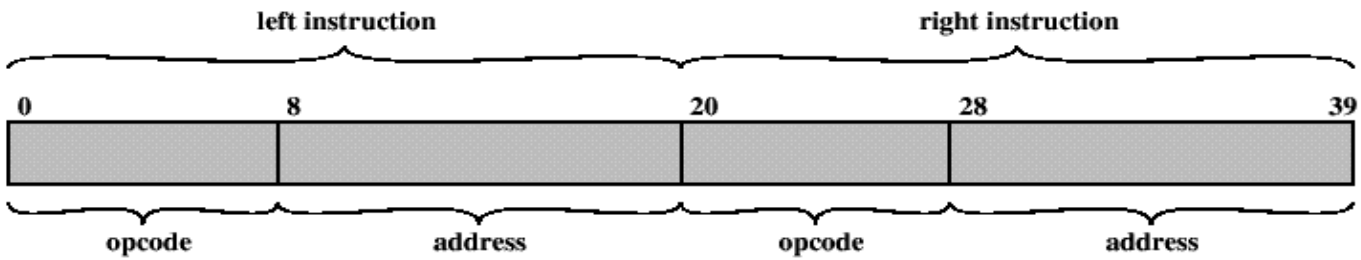


Memory of the IAS

- 1000 storage locations called words.
- Each word 40 bits.
- A word may contain:
  - A numbers stored as 40 binary digits (bits) – sign bit + 39 bit value
  - An instruction-pair. Each instruction:
    - An opcode (8 bits)
    - An address (12 bits) – designating one of the 1000 words in memory.



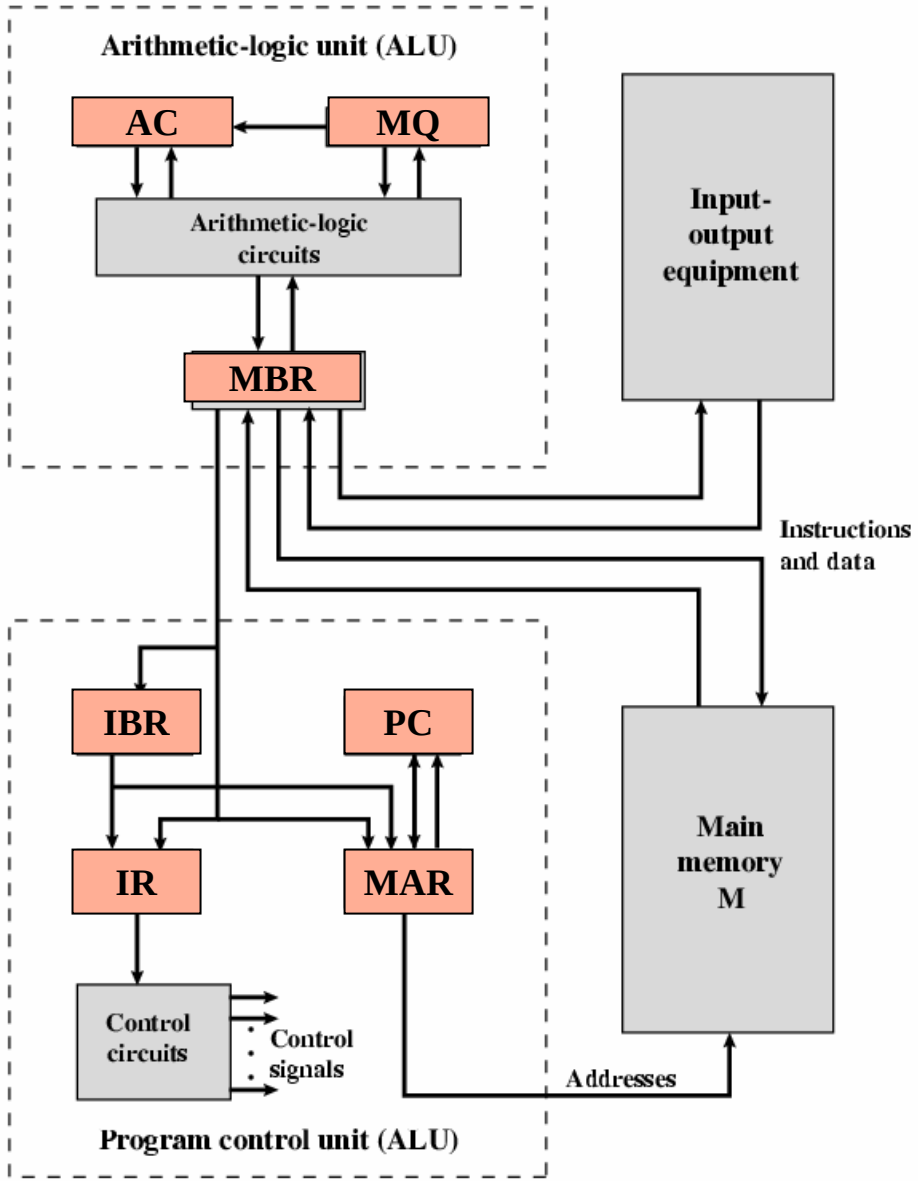
(a) Number word



(b) Instruction word

Von Neumann Machine

- **MBR: Memory Buffer Register**
  - contains the word to be stored in memory or just received from memory.
- **MAR: Memory Address Register**
  - specifies the address in memory of the word to be stored or retrieved.
- **IR: Instruction Register -** contains the 8-bit opcode currently being executed.
- **IBR: Instruction Buffer Register**
  - temporary store for RHS instruction from word in memory.
- **PC: Program Counter -** address of next instruction-pair to fetch from memory.
- **AC: Accumulator & MQ: Multiplier quotient** - holds operands and results of ALU ops.



IAS Instruction set

Instruction Type	Opcode	Symbolic Representation	Description
Data transfer	00001010	LOAD MQ	Transfer contents of register MQ to the accumulator AC
	00001001	LOAD MQ,M(X)	Transfer contents of memory location X to MQ
	00100001	STOR M(X)	Transfer contents of accumulator to memory location X
	00000001	LOAD M(X)	Transfer M(X) to the accumulator
	00000010	LOAD –M(X)	Transfer –M(X) to the accumulator
	00000011	LOAD |M(X)|	Transfer absolute value of M(X) to the accumulator
	00000100	LOAD –|M(X)|	Transfer –|M(X)| to the accumulator
Unconditional branch	00001101	JUMP M(X,0:19)	Take next instruction from left half of M(X)
	00001110	JUMP M(X,20:39)	Take next instruction from right half of M(X)
Conditional branch	00001111	JUMP+ M(X,0:19)	If number in the accumulator is nonnegative, take next instruction from left half of M(X)
	00010000	JUMP+ M(X,20:39)	If number in the accumulator is nonnegative, take next instruction from right half of M(X)

IAS Instruction set (continued)

Arithmetic	00000101	ADD M(X)	Add M(X) to AC; put the result in AC
	00000111	ADD |M(X)|	Add |M(X)| to AC; put the result in AC
	00000110	SUB M(X)	Subtract M(X) from AC; put the result in AC
	00001000	SUB |M(X)|	Subtract |M(X)| from AC; put the remainder in AC
	00001011	MUL M(X)	Multiply M(X) by MQ; put most significant bits of result in AC, put least significant bits in MQ
	00001100	DIV M(X)	Divide AC by M(X); put the quotient in MQ and the remainder in AC
	00010100	LSH	Multiply accumulator by 2, i.e., shift left one bit position
	00010101	RSH	Divide accumulator by 2, i.e., shift right one position
Address modify	00010010	STOR M(X,8:19)	Replace left address field at M(X) by 12 rightmost bits of AC
	00010011	STOR M(X,28:39)	Replace right address field at M(X) by 12 rightmost bits of AC

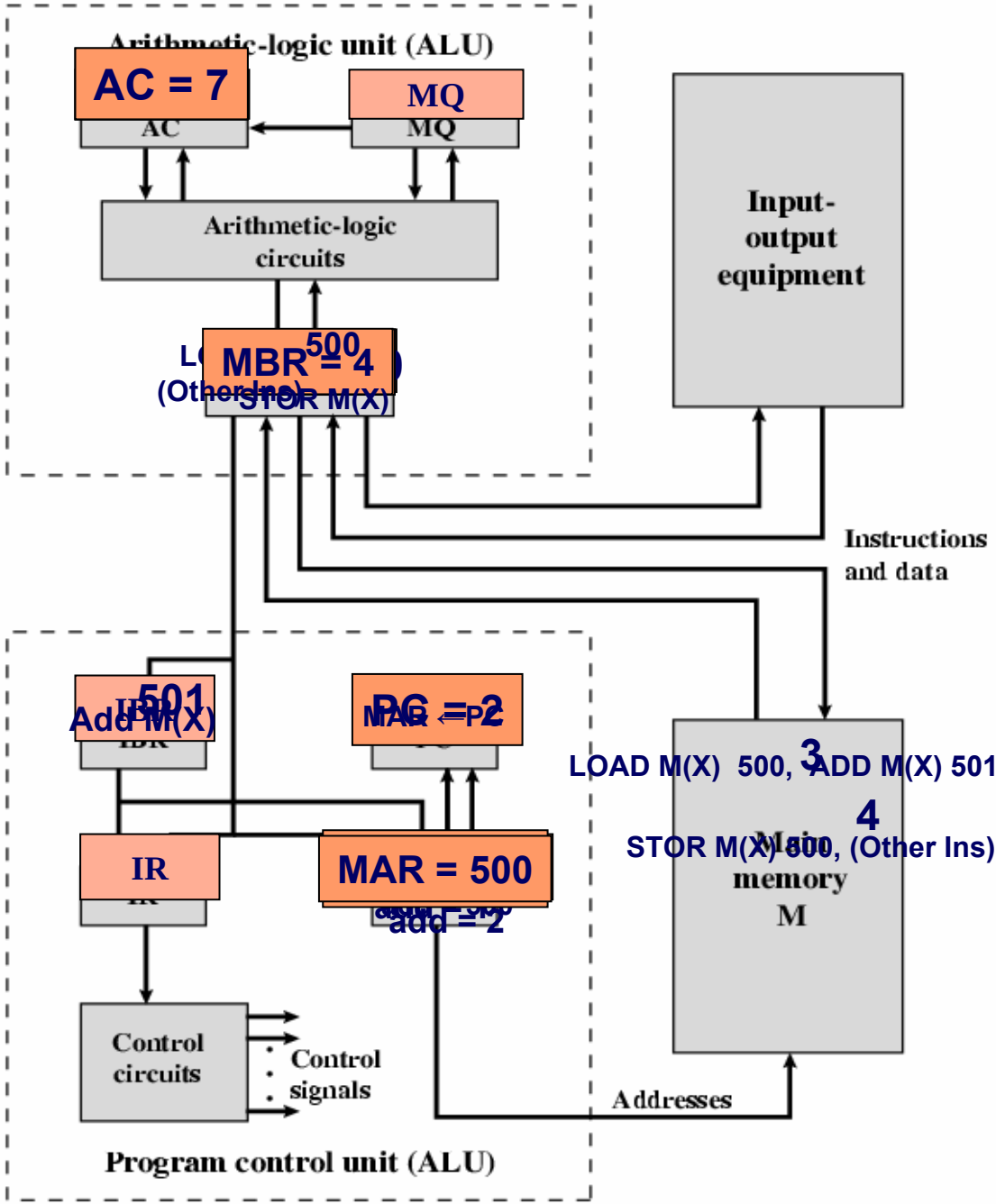
Example of an Instruction-pair.

Load M(100), Add M(101)

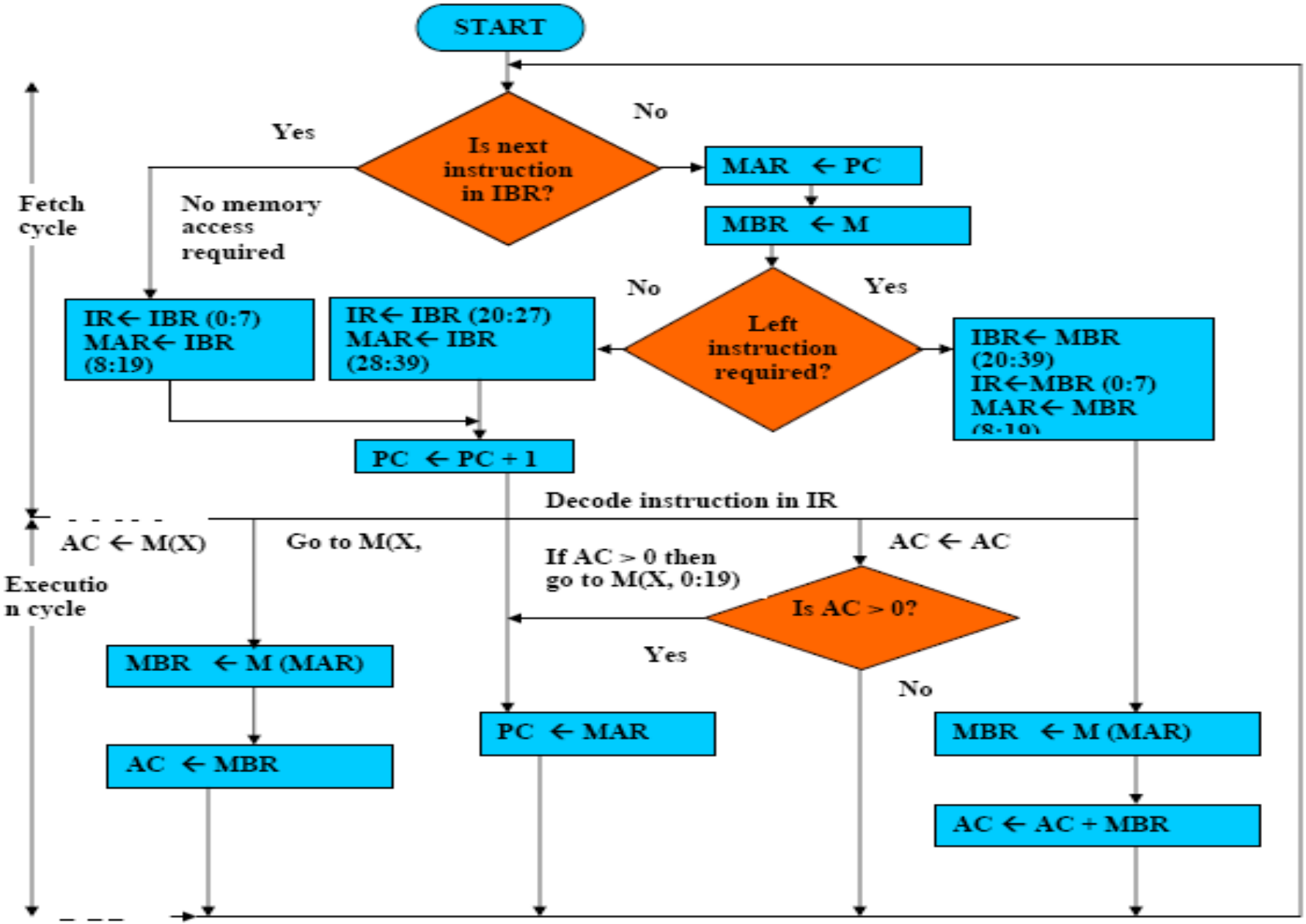
00000001000111110100000010100011111010 1

MEMORY
1. LOAD M(X) 500, ADD M(X) 501
2. STOR M(X) 500, (Other Ins)
.....
500. 3
501. 4

PC	2
MAR	500
MBR	STOR M(X) 500, (Other Ins)
IR	STOR M(X)
IBR	(Other Ins)
AC	7



Fetch / Execute Cycle



INSTRUCTION FORMAT

Instruction Fields

OP-code field - specifies the operation to be performed
Address field - designates memory address(s) or a processor register(s)
Mode field - specifies the way the operand or the effective address is determined

The number of address fields in the instruction format depends on the internal organization of CPU

- The three most common CPU organizations:

Single accumulator organization:

```
ADD     X                   /* AC ← AC + M[X]  */
```

General register organization:

```
ADD     R1, R2, R3          /* R1 ← R2 + R3  */
ADD     R1, R2              /* R1 ← R1 + R2  */
MOV     R1, R2              /* R1 ← R2  */
ADD     R1, X               /* R1 ← R1 + M[X]  */
```

Stack organization:

```
PUSH    X                   /* TOS ← M[X]  */
ADD
```

THREE, and TWO-ADDRESS INSTRUCTIONS

Three-Address Instructions:

Program to evaluate X = (A + B) * (C + D) :

```
ADD    R1, A, B        /*  R1 ← M[A] + M[B]        */
ADD    R2, C, D        /*  R2 ← M[C] + M[D]        */
MUL    X, R1, R2       /*  M[X] ← R1 * R2          */
```

- Results in short programs
- Instruction becomes long (many bits)

Two-Address Instructions:

Program to evaluate X = (A + B) * (C + D) :

```
MOV    R1, A           /* R1 ← M[A]          */
ADD    R1, B           /* R1 ← R1 + M[B]     */
MOV    R2, C           /* R2 ← M[C]          */
ADD    R2, D           /* R2 ← R2 + M[D]     */
MUL    R1, R2          /* R1 ← R1 * R2       */
MOV    X, R1           /* M[X] ← R1          */
```

ONE, and ZERO-ADDRESS INSTRUCTIONS

One-Address Instructions:

- Use an implied AC register for all data manipulation
- Program to evaluate X = (A + B) * (C + D) :

```
LOAD    A    /*  AC ← M[A]            */
ADD     B    /* AC ← AC + M[B]       */
STORE   T    /*  M[T] ← AC           */
LOAD    C    /*  AC ← M[C]           */
ADD     D    /*  AC ← AC + M[D]      */
MUL     T    /*  AC ← AC * M[T]      */
STORE   X    /*  M[X] ← AC           */
```

Zero-Address Instructions:

- Can be found in a stack-organized computer
- Program to evaluate X = (A + B) * (C + D) :

```
PUSH    A    /*  TOS ← A             */
PUSH    B    /*  TOS ← B             */
ADD          /*  TOS ← (A + B)       */
PUSH    C    /*  TOS ← C             */
PUSH    D    /*  TOS ← D             */
ADD          /*  TOS ← (C + D)       */
MUL          /*  TOS ← (C + D) * (A + B)  */
POP     X    /*  M[X] ← TOS          */
```

RISC

- Reduced inst set computer
- A pgm for RISC type CPU consist of LOAD and STORE inst that have one memory and one register address and computational type inst that have 3 address with all 3 specifying processor reg.

ADDRESSING MODES

Addressing Modes:

* Specifies a rule for interpreting or modifying the address field of the instruction (before the operand is actually referenced)

* Variety of addressing modes

  - to give programming flexibility to the user
  - to use the bits in the address field of the instruction efficiently

Types of addressing modes

- Implied
- Immediate
- Register
- Register Indirect
- Auto Increment or Auto Decrement
- Direct
- Indirect
- Relative
- Index
- Base Register

- **Implied:-**
- No address field
- Operands are specified implicitly in the definition of instruction.
- EX: Complement accumulator CMA,INC,DEC
- Zero address inst in stack organization.
- **Immediate:-**
- An immediate mode inst has an operand field rather than an address field.
- Operand field contains the actual operand to be used in conjunction with the operation specified in the inst
- Mov R1,#200

- **Register Address mode:-**
- Operand are in registers that reside within the CPU.
- Ex: MOV R1,R2
- **Register Indirect address:-**
- In the mode the inst specifies a register in the CPU whose content give the address of the operand in memory.
- Eg: MOV R1,(R2)

- **Auto increment or Auto decrement:-**
- Similar to register indirect mode except that the register is incremented or decrement after its value is used to access memory.
- 2 forms **post** and **pre :-**
- **Mov R1,(R2)+ → post increment**
- **Mov R1,+(R2) → pre increment**
- **Mov R1,(R2)- → post decrement**
- **Mov R1,-(R2) → post decrement**

Direct :-

- Address field contains the direct address of the operands.
- MOV R1,2000

Indirect address mode:

- the address field of the inst gives address where the effective address is stored in memory.

Effective address:-

- memory address obtained from the computation dictated by the given addressing mode. It is the address of the operand in a computational type inst
- MOV R1,(2000)
- Mov 20(R1),R2

Relative:-

- Program counter value +address field → effective address
- LOAD R1,$ADR.

Index:-

- **In** this mode the content of an index register is added to the address part of inst to obtain the effective address.
- LOAD ADR(x)

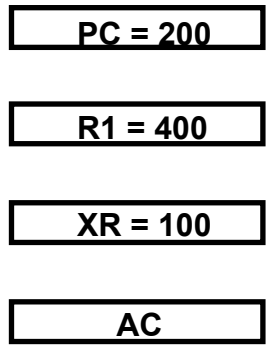
Base Reg:-

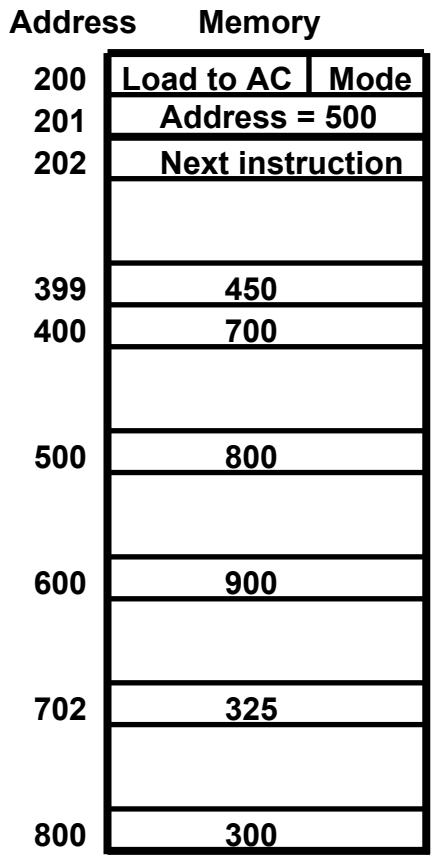
Base Reg +Address field→ effective address

Index Reg→ holds the index number

Base Reg→holds the base address & address field gives the displacement.

ADDRESSING MODES - EXAMPLES





Addressing Mode	Effective Address			Content of AC
Direct address	500	/* AC ← (500)	*/	800
Immediate operand	-	/* AC ← 500	*/	500
Indirect address	800	/* AC ← ((500))	*/	300
Relative address	702	/* AC ← (PC+500)	*/	325
Indexed address	600	/* AC ← (XR+500)	*/	900
Register	-	/* AC ← R1	*/	400
Register indirect	400	/* AC ← (R1)	*/	700
Autoincrement	400	/* AC ← (R1)+	*/	700
Autodecrement	399	/* AC ← -(R)	*/	450

Instruction types

DATA TRANSFER INSTRUCTIONS

Typical Data Transfer Instructions

Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

Data Transfer Instructions with Different Addressing Modes

Mode	Assembly Convention	Register Transfer
Direct address	LD ADR	AC ← M[ADR]
Indirect address	LD @ADR	AC ← M[M[ADR]]
Relative address	LD $ADR	AC ← M[PC + ADR]
Immediate operand	LD #NBR	AC ← NBR
Index addressing	LD ADR(X)	AC ← M[ADR + XR]
Register	LD R1	AC ← R1
Register indirect	LD (R1)	AC ← M[R1]
Autoincrement	LD (R1)+	AC ← M[R1], R1 ← R1 + 1
Autodecrement	LD -(R1)	R1 ← R1 - 1, AC ← M[R1]

DATA MANIPULATION INSTRUCTIONS

Three Basic Types: **Arithmetic instructions**
Logical and bit manipulation instructions
Shift instructions

Arithmetic Instructions

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with Carry	ADDC
Subtract with Borrow	SUBB
Negate(2's Complement)	NEG

Logical and Bit Manipulation Instructions

Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC
Enable interrupt	EI
Disable interrupt	DI

Shift Instructions

Name	Mnemonic
Logical shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate right thru carry	RORC
Rotate left thru carry	ROLC

Subroutine Call and Return Mechanisms

- Subroutine is a self-contained sequence of instructions that performs a given computational task.
- It may be called many times at various points in the main program
- When called, branches to 1st line of subroutine and at the end, returned to main program.
- Different names to the instruction that transfers program control to a subroutine
  - Call subroutine
  - Jump to subroutine
  - Branch to subroutine
  - Branch and save address

Trace the given program

```
Void Main()                    //Main() is the faculty
{
    say good morning;
    Write “Introduction”;
    call “student1”;
    Write “Thankyou”;
}
```

Continue Tracing

```
Void student1()
{
   say nursery rhyme;
   Write “Computer Architecture”;
   call “student2”;
   Write your School name;
}
```

Continue Tracing

```
Void student2()
{
   act like vegetable vendor;
   Write “B1 slot”;
   call “student3”;
   Write your branch;
}
```

Continue Tracing

```
Void student3()
{
Sing a song;
Write your name;
}
```

Subroutine call

- CALL

  SP ← SP – 1 decrement SP

  M[SP] ← PC Push content of PC into stack

  PC ← Effective Address (EA) Transfer control to subroutine.

- RTN

  PC ← M[SP] POP stack and transfer to PC.

  SP ← SP + 1 Inc SP

Recursive subroutine is a subroutine that calls itself.

Recursive subroutines

- Subroutine that calls itself
- If only one register or memory location is used to hold the return address, when subroutine is called recursively, it destroys the previous return address.
- So, stack is the good solution for this problem

Instruction fetch and Execute

- The fetched instruction is loaded into a register in the processor is called as instruction register(IR).
- Category of Actions :

1. Processor memory:

   Data may be transferred from processor to memory or from memory to processor.

Cont..

2. Processor I/O:

   Data may be transferred to or from a peripheral device by transferring between the processor and I/O module.

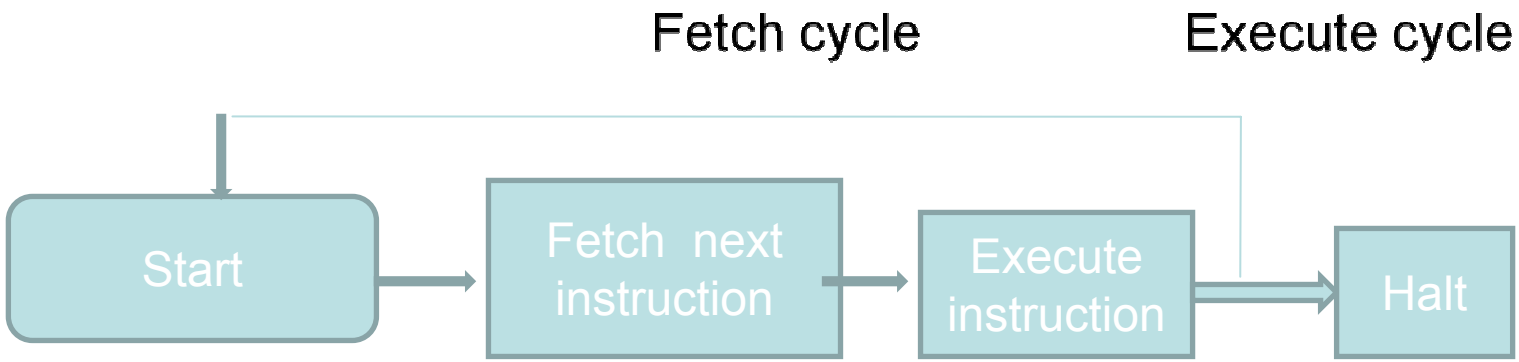
3. Data processing:

   The processor may perform some arithmetic or logic operation on data.

4. control:

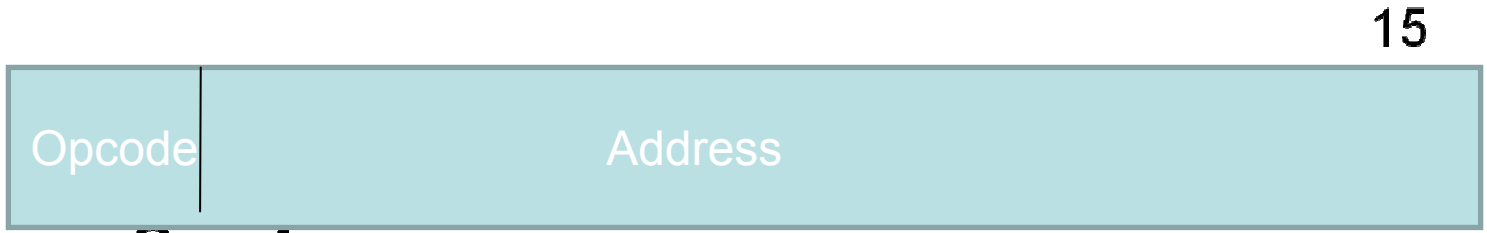
   An instruction may specify the sequence of execution be altered.

Basic Instruction cycle

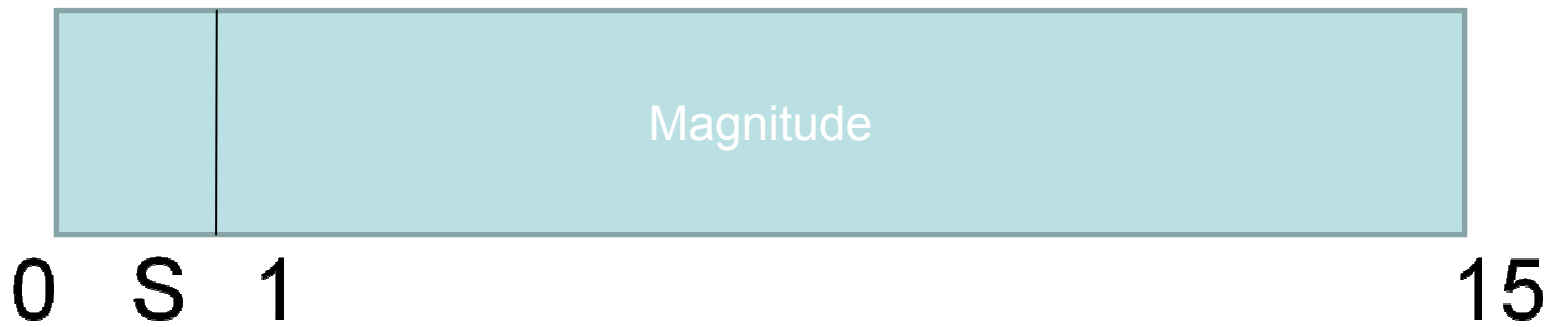


1. Instruction format

»



- 2.Integer format



Cont..

- 3. Internal CPU registers:

  Program counter(PC)- Address of instruction

Instruction Register (IR)- Instruction being executed

Accumulator(AC)- Temporary register.

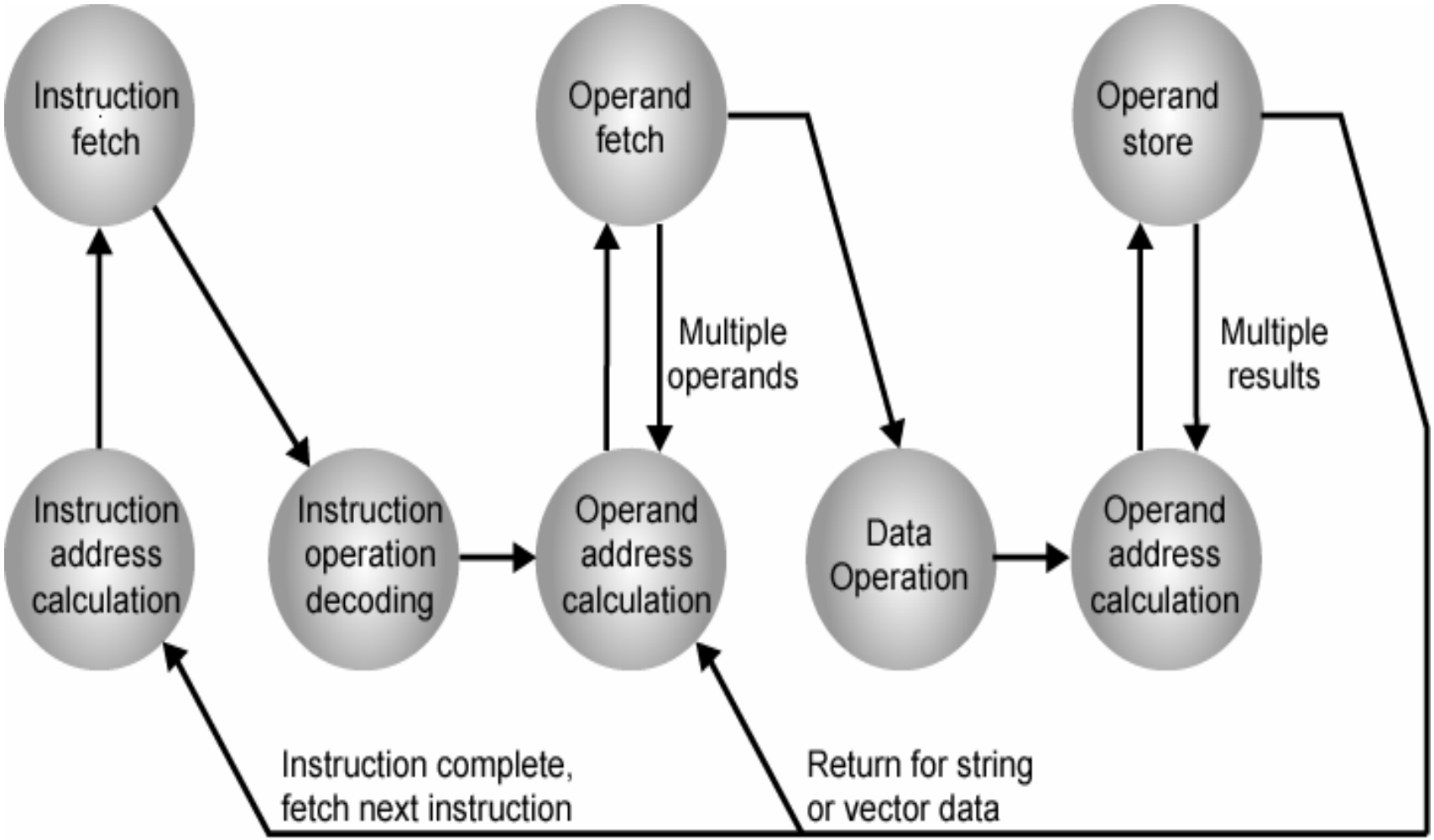
Partial list of opcodes:

0001- Load AC from memory

0010- Store AC from memory

0101- Add to AC from memory

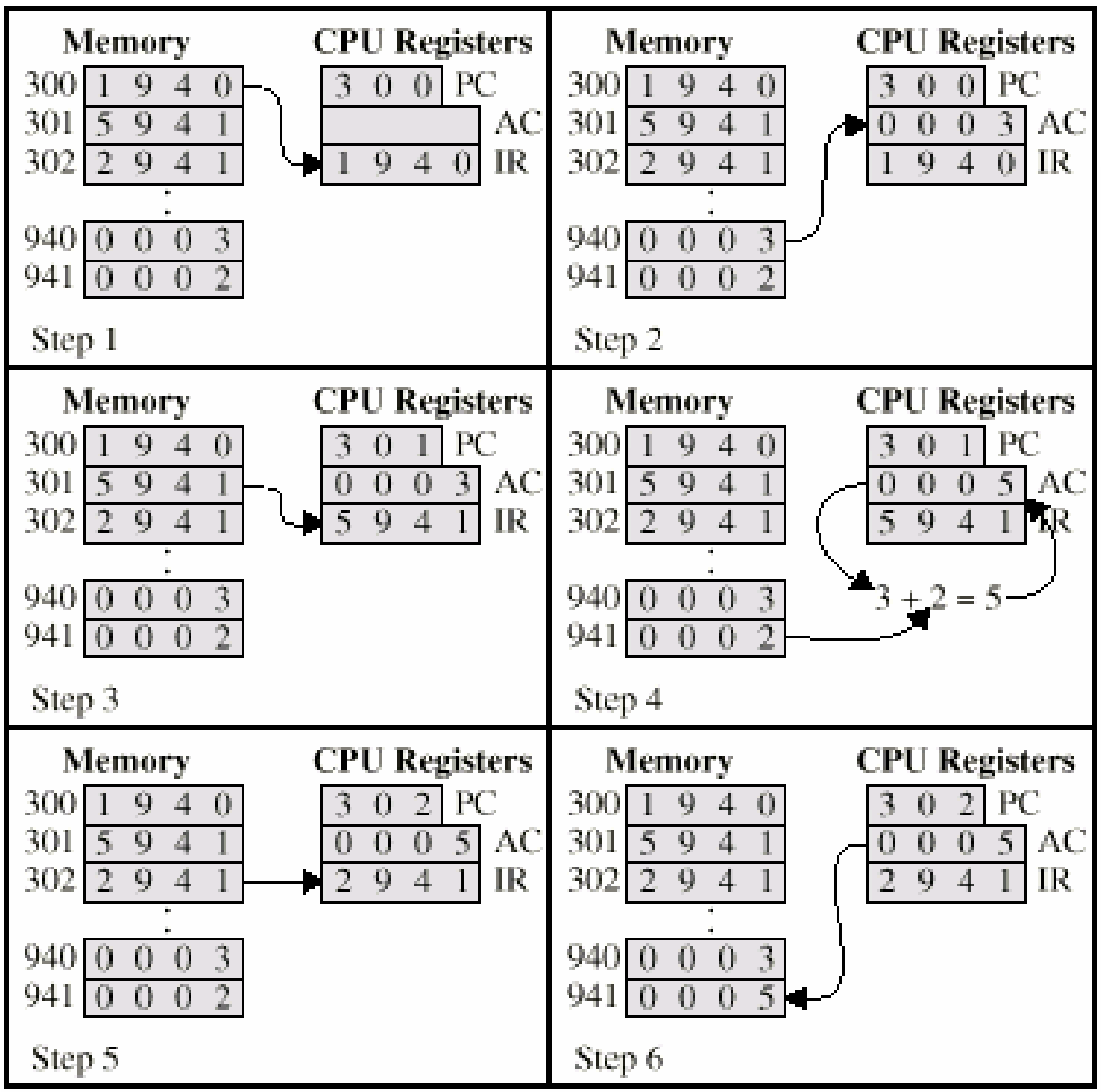
Instruction Cycle State Diagram



Instruction state cycle diagram (cont..)

- Instruction address calculation (iac):
  - Determine the address of the next instruction to be executed. Adding a fixed number to a next number.
- Instruction fetch: (if)
  - Read the instruction from its memory location into the processor.
- Instruction operation decoding: (iad)
  - Analyze instruction to determine type of operation to be performed and operand(s) to be used.
- Operand Address Calculation: (oac)
  - If the operation involves the reference to an operand in memory or available via I/O, then determine the address of the operand.
- Operand Fetch (of):
  - Fetch the operand from memory or read it from I/O.
- Data Operation (do):
  - Perform the operation indicated in the instruction.
- Operand store (os)
  - Write the result into memory or out to I/O.

Example of Program Execution



0001 = Load AC from Memory
0010 = Store AC to Memory
0101 = Add to AC from Memory

Program interrupt

- Program interrupt refers to transfer of program ctrl form currently running program to another service program as a result of external or internal generated request.
- Interrupt is usually initiated by an internal or external signal rather from execution of inst.
- Address of ISR (Interrupt Service Program) is determined by H/W rather from address field.
- Interrupt procedure usually stores all the information necessary to define the state of CPU rather than storing only PC.

Cont..

- PSW (program status word)
- Collection of all status bit conditions in the CPU is called PSW.
- It is a hardware register and contains status information that characterizes state of CPU.

PROGRAM INTERRUPT

Types of Interrupts:

External interrupts

External Interrupts initiated from the outside of CPU and Memory
- I/O Device -> Data transfer request or Data transfer complete
- Timing Device -> Timeout
- Power Failure

Internal interrupts (trap)

Internal Interrupts are caused by the currently running program
- Register, Stack Overflow
- Divide by zero
- OP-code Violation
- Protection Violation

Software Interrupts

Both External and Internal Interrupts are initiated by the computer Hardware.
Software Interrupts are initiated by executing an instruction.
- Supervisor Call -> Switching from a user mode to the supervisor mode
  -> Allows to execute a certain class of operations
  which are not allowed in the user mode

Register and Register Files

Cont..

- An m- bit register is an ordered set of m flip flops designed to store m word. (z0… zm-1)
- Each bit is stored in separate flip flops.
- All flip flops have common control line.(clk, clear)
- Registers can be made up of various flip flops.
- diagram

Cont…

- Register z reads in the data word x each time it is clocked. To maintain the contents of z constant, z inputs bus should be enabled always.
- Parallel load: diagram

  introduce a control line load, which causes the register to read in the current value of x when it is clocked and load=1, if load=0 the previous value is retained in the register.

Shift register

- Transfer the contents into and from register 1bit at a time.
- M- bit shift register consists of m flip flops each of which is connected to its left or right neighbor.
- Data can be entered 1 bit at a time at one end of register and can be removed 1 bit at a time

  from the other end known as serial i/p –o/p.

diagram

Cont..

- Right shift is accomplished by activating the shift enable line connected to the clk input of each flip flop.
- Right shift operation changes the registers state as,
- (Xzm-1 zm-2…z1):=(zm-1 zm-2,….z1,z0)
- Left shift operation changes the register state as (zm-2 zm-3…z0,x):=(zm-1 zm-2,….z1,z0)

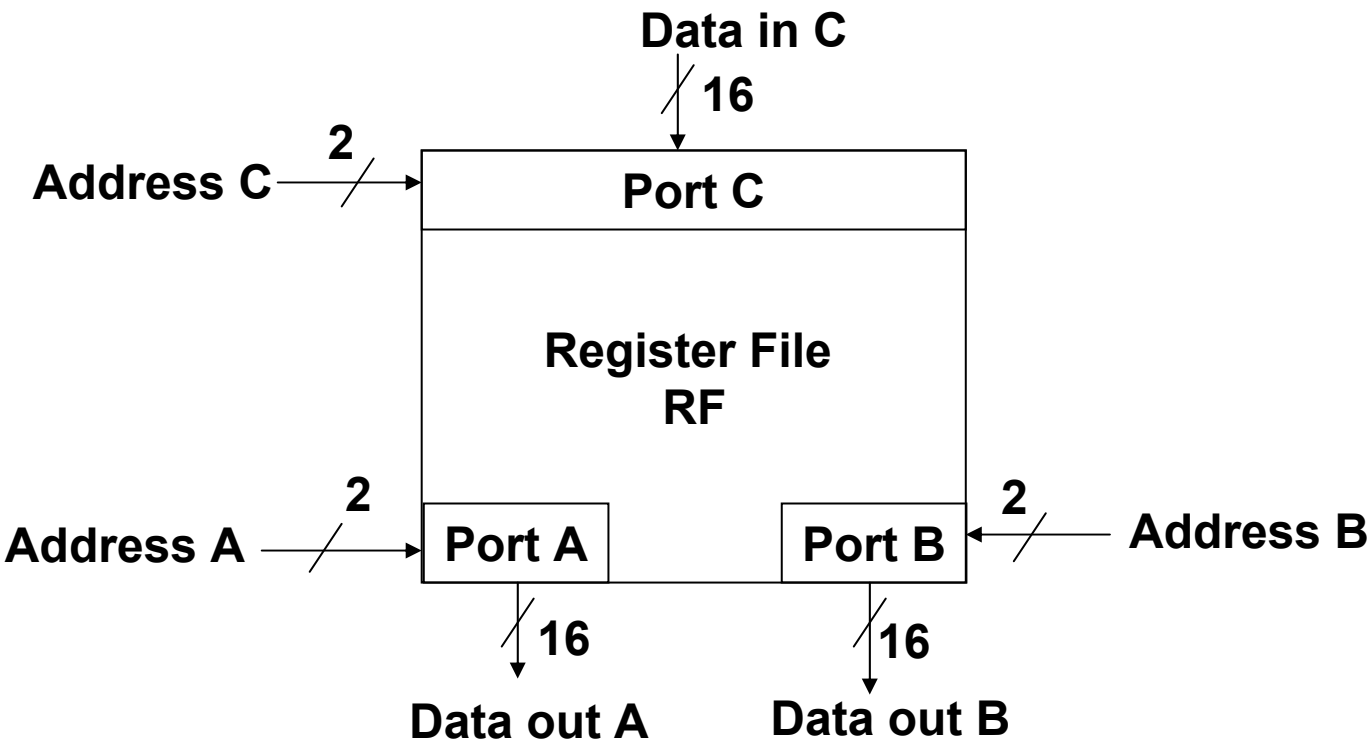
Register files

- Set of general purpose registers r0:rm-1 known as register file RF.
- Each register Ri is individually addressable.
- Arithmetic logic instruction take two ,three address format.
- R2=f(R1,R2)- 2 address
- R3=f(R1,R2)- 3 address

Cont..

- Intermediate results are stored in processor with the help of register.
- RF functions as a RAM.
- RF differs from M becomes RF requires 2 or more operands accessed simultaneously.
- RF is also known as multiport RAM

A register file with three access ports - symbol



A Register File with three access ports – logic diagram