Computer Architecture and Microprocessor

Ass.Prof.Shaimaa Hameed Shaker

References

1. **"fundamentals of computer organization and architecture**", by Mostafa Abd-El-Barr Hesham El-Rewini, 1PstP edition/2005.
2. David A. Patterson and John L. Hennessy, **"Computer Organization and Design"**,1998.
3. M. M. Mano, **"computer system architecture"** third edition, prentice Hall, 1993.
4. Walter A. Triebel, **"The 80386, 80486, and Pentium® Processors Hardware,Software, and Interfaceing",** 1998.

Control Unit

- **Initiate sequences of microoperations**

  - **Control signal** *(that specify microoperations)*in a bus-organized system

    groups of bits that select the paths in multiplexers, decoders, and arithmetic logic units

- **Two major types of Control Unit**

  - Hardwired Control :
    - The control logic is implemented with gates, F/Fs, decoders, andother digital circuits
    - \+ Fast operation.
    - –Wiring change(if the design has to be modified)

  - Microprogrammed Control
    - The control information is stored in a control memory, and the control memory is programmed to initiate the required sequence of microoperations
    - +Any required change can be done by updating the microprogram incontrol memory.
    - –Slow operation

Introduction

The performance of a program depends on a combination of the effectiveness of the algorithms used in the program, the software systems used to create and translate the program into machine instructions, and the effectiveness of the computer in executing those instructions, which may include input/output (I/O) operations.

positions 12 through 15.In the 8086, these bits are stored as ones, but in 80386 real-address mode bit 15 is always zero, and bits 12 through 14 reflect the last value loaded into them.

Microprogrammed Control Unit

Control Word

The control variables at any given time can be represented by a string of 1's and 0's.

Microprogrammed Control Unit

A control unit whose binary control variables are stored in memory (*control memory*).

Microinstruction

The microinstruction specifies one or more microoperations

Microprogram

A sequence of microinstruction

- Dynamic microprogramming :*Control Memory* =RAM
  - RAM can be used for writing (*to change a writable control memory*)
  - Microprogram is loaded initially from an auxiliary memory such as a magnetic disk
- Static microprogramming : *Control Memory* =ROM
  - Control words in ROM are made permanent during the hardware production.

Hardwired andMicroprogrammed Control

For each instruction, the control unit causes the CPU to execute a sequence ofsteps correctly. In reality, there must be control signals to assert lines on various digitalcomponents to make things happen. For example, when we perform an Add instruction inassembly language, we assume the addition takes place because the control signals forthe ALU are set to "add" and the result is put into the AC. The ALU has various controllines that determine which operation to perform. The question we need to answer is,"How do these control lines actually become asserted?" We can take one of two approaches to ensure control lines are set properly. Thefirst approach is to physically connect all of the control lines to the actual machineinstructions. The instructions are divided up into fields, and different bits in theinstruction are combined through various digital logic components to drive the controllines. This is called hardwired control, and is illustrated in figure (1)

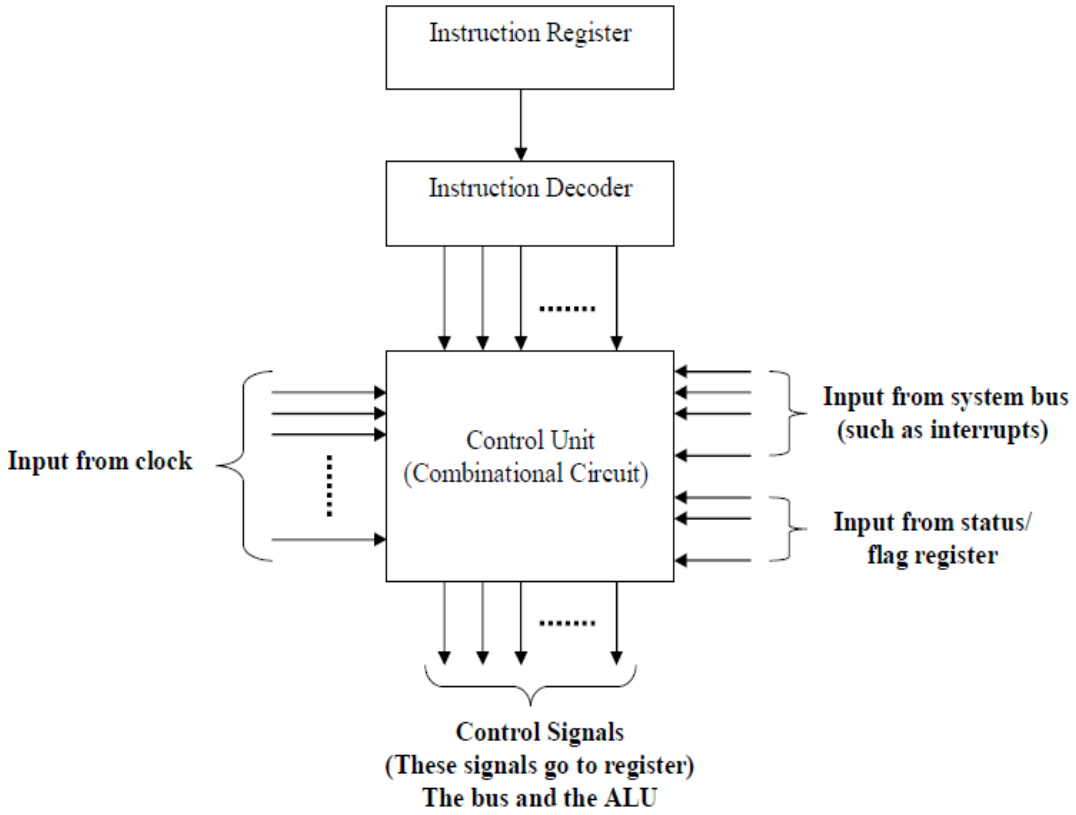


Figure (1) Hardwired Control Organization

The control unit is implemented using hardware (for example:NANDgates, flip-flops, and counters).We need a special digital circuit that uses , as inputs, thebits from the Opcode field in our instructions, bits from the flag (or status) register,signals from the bus, and signals from the clock. It should produce, as outputs, thecontrol signals to drive the various components in the computer.

The advantage of hardwired control is that is very fast. The disadvantage is that theinstruction set and the control logic are directly tied together by special circuits that arecomplex and difficult to design or modify. If someone designs a hardwired computerand later decides to extend the instruction set, the physical components in the computermust be changed. This is prohibitively expensive, because not only must new chips befabricated but also the old ones must be located and replaced.

Microprogramming is a second alternative for designing control unit of digitalcomputer (uses software for control). A control unit whose binary control variables arestored in memory is called a **microprogrammed control unit**. The control variables atany given time can be represented by a string of 1's and 0's called a **control word** (whichcan be programmed to perform various operations on the component of the system). Eachword in control memory contains within it a

microinstruction. The microinstructionspecifies one or more **microoperatiotins**for the system. A sequence of microinstructions constitutes a **microprogram.** A memory that is part of a control unit is referred to as a**control memory.**A more advanced development known as **dynamic** microprogramming permits amicroprogram to be loaded initially from an auxiliary memory such as a magnetic disk.Control units that use dynamic microprogramming employ a writable control memory; this type of memory can be used for writing (to change the microprogram) but is used mostly for reading.

The general configuration of a microprogrammed control unit is demonstrated inthe block diagram of Figure (2). The control memory is assumed to be a ROM, withinwhich all control information is permanently stored.

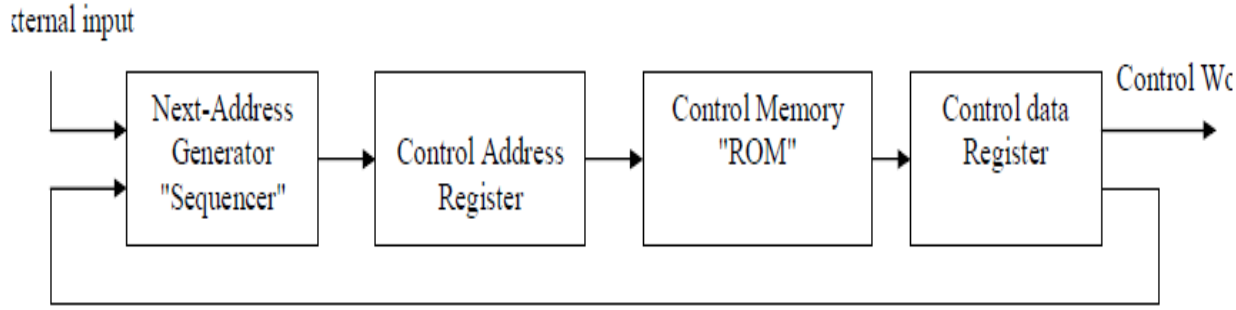


Figure (2) Microprogrammed Control Organization

The control memory address register specifies the address of the microinstruction andthe control data register holds the microinstruction read from memory themicroinstruction contains a control word that specifies one or more microoperations forthe data processor. Once these operations are executed, the control must determine thenext address. The location of the next microinstruction may be the one next in sequence, or it may be locate somewhere else in the control memory. For this reason it is necessaryto use some bits of the present microinstruction to control the generation of the address ofthe next microinstruction. The next address may also be a function of external inputconditions. While the microoperations are being executed, the next address is computedin the next address generator circuit and then transferred into the control address registerto read the next microinstruction.The next address generator is sometimes called a microprogram sequencer,as it determines the address sequence that is read from control memory, the address of the

next microinstruction can be specified several ways, depending on the sequencerinputs. Typical functions of a microprogram sequencer are incrementing the controladdress register by one, loading into the control address register an address from controlmemory, transferring an external address or loading an initial address to start the controloperations.

The main advantages of the microprogrammed control are the fact that once thehardware configuration is established; there should be no need for further hardware orwiring changes. If we want to establish are different control sequence for the system, allwe need to do is specify different set microinstructions for control memory. Thehardware configuration should not be changed for different operations; the only thing thatmust be changed is the microprogram residing in control memory.Microinstructions are stored in control memory in groups, with each groupspecifying routine. Each computer instruction has microprogram routine in controlmemory to generate the microoperations that execute the instruction. The hardware thatcontrols the address sequencing of the control memory must be capable of sequencing the microinstructions within a routine and be to branch from one routine to another. Theaddress sequencing capabilities required in a control memory are:

1. Incrementing of the control address register.
2. Unconditional branch or conditional branch, depending on status bit conditions.
3. A mapping process from the bits of the instruction to an address for control memory.
4. A facility for subroutine call and return.

Figure (3) shows a block diagram of control memory and the associated hardware needed for selecting the next microinstruction address. The microinstruction in controlmemory contains a set of bits to initiate microoperations in computer registers and otherbits to specify the method by which the address is obtained. The diagram shows four different paths from which the control address register (CAR) receives the address. Theincrementer increments the content of the control address register by one, to select thenext microinstruction in sequence. Branching is achieved by specifying the branchaddress in one of the fields of the microinstruction. Conditional branching is obtained by using part of the microinstruction to select a specific status bit in order to determine itscondition. An external address is transferred into control memory via a **mapping logic**circuit. The return address for a subroutine is stored in a special register whose value isthen used when the microprogram wishes to return from the subroutine.

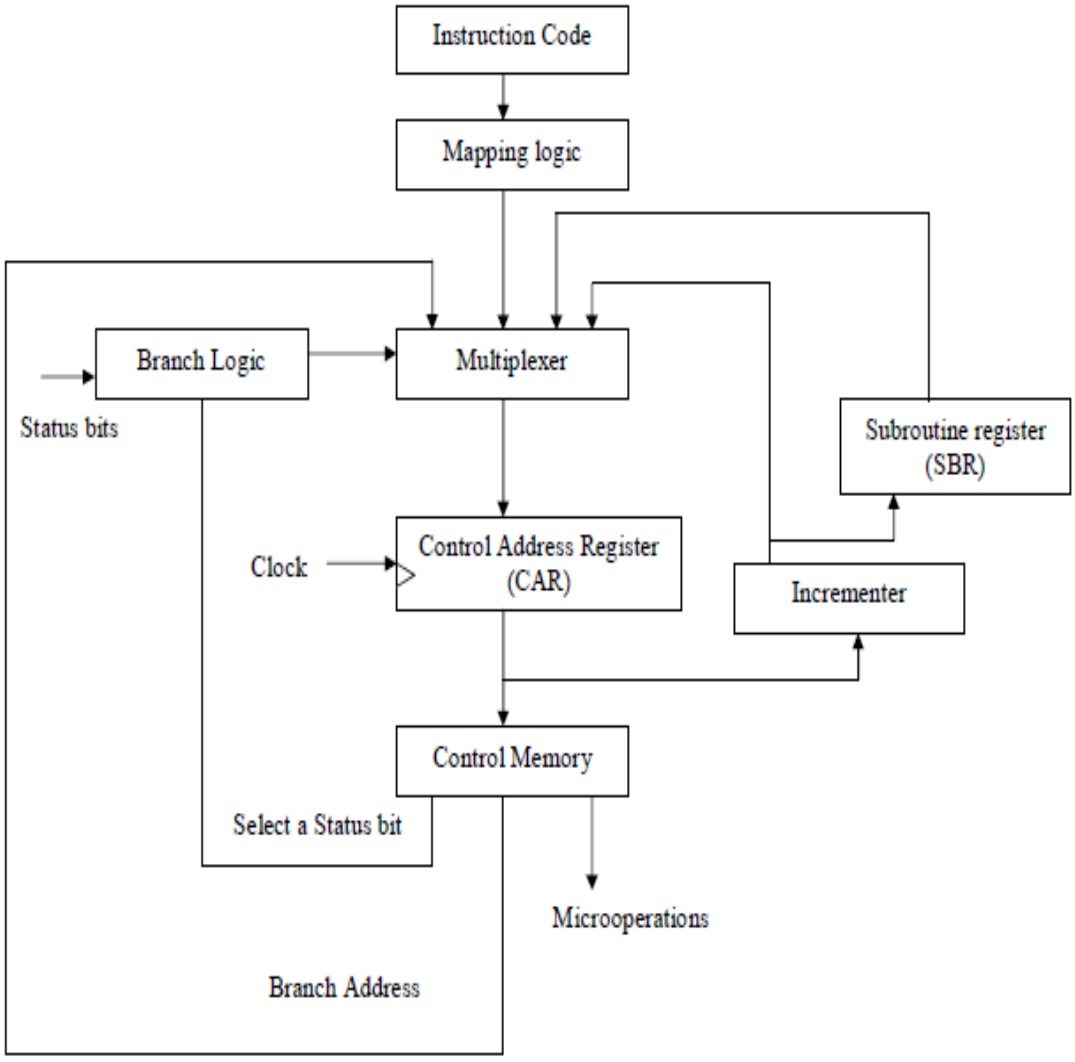
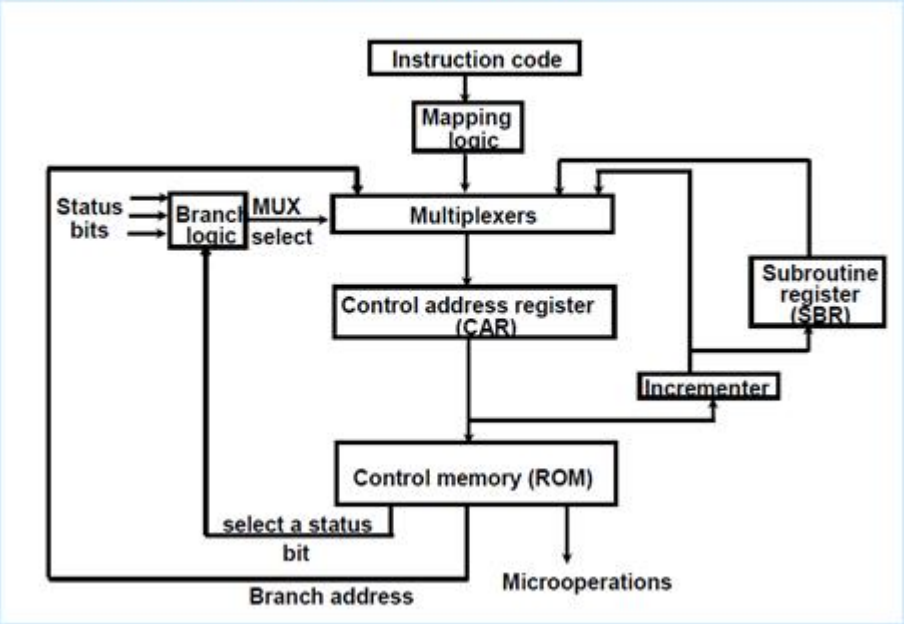


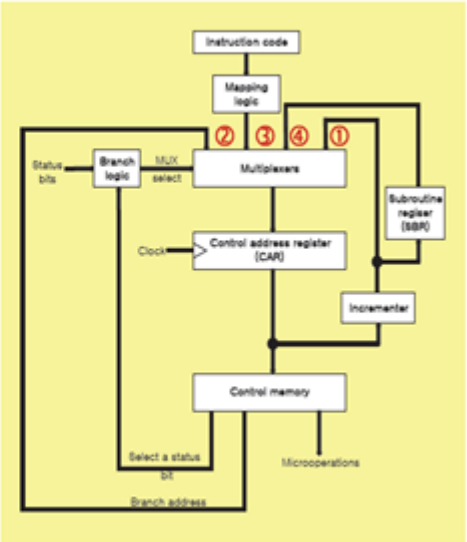
Figure (3) Selection address for control memory

Selection of address for control memory

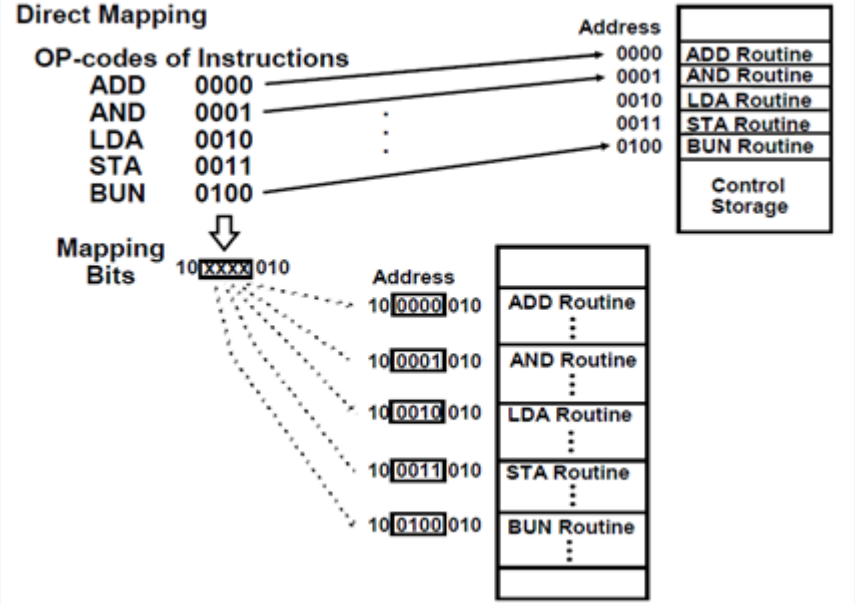


◆ Selection of address for control memory : *Fig. 7-2*

- Multiplexer
  - ① CAR Increment
  - ② JMP/CALL
  - ③ Mapping
  - ④ Subroutine Return
- CAR : Control Address Register
  - » CAR receive the address from 4 different paths
    1) Incrementer
    2) Branch address from control memory
    3) Mapping Logic
    4) SBR : Subroutine Register
- SBR : Subroutine Register
  - » Return Address can not be stored in ROM
  - » Return Address for a subroutine is stored in SBR

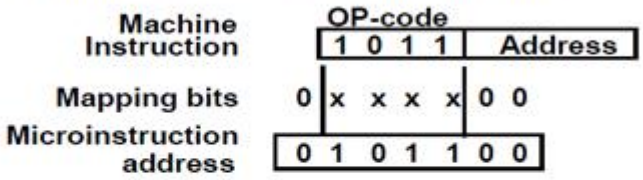


Mapping of Instructions

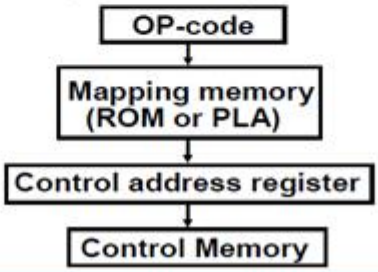


Mapping of Instructions to Microroutines

Mapping from the OP-code of an instruction to the address of the Microinstruction which is the starting microinstruction of its execution microprogram



Mapping function implemented by ROM or PLA



Central Processing Unit

MAJOR COMPONENTS OF CPU
-Storage Components

- **Registers**
- **Flags**

-Execution(Processing) Components

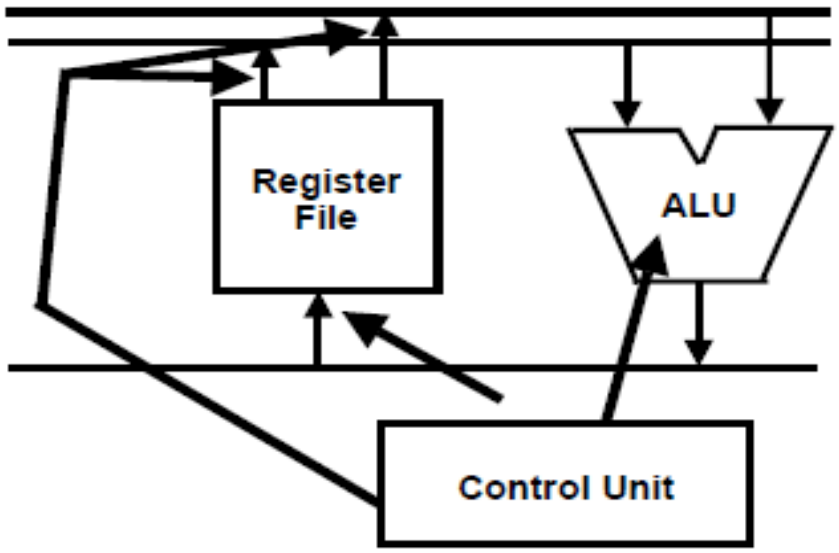
- **Arithmetic Logic Unit(ALU)**

Arithmetic calculations, Logical computations, Shifts/Rotates
-Transfer Components

- **Bus**

-Control Components

- **Control Unit**



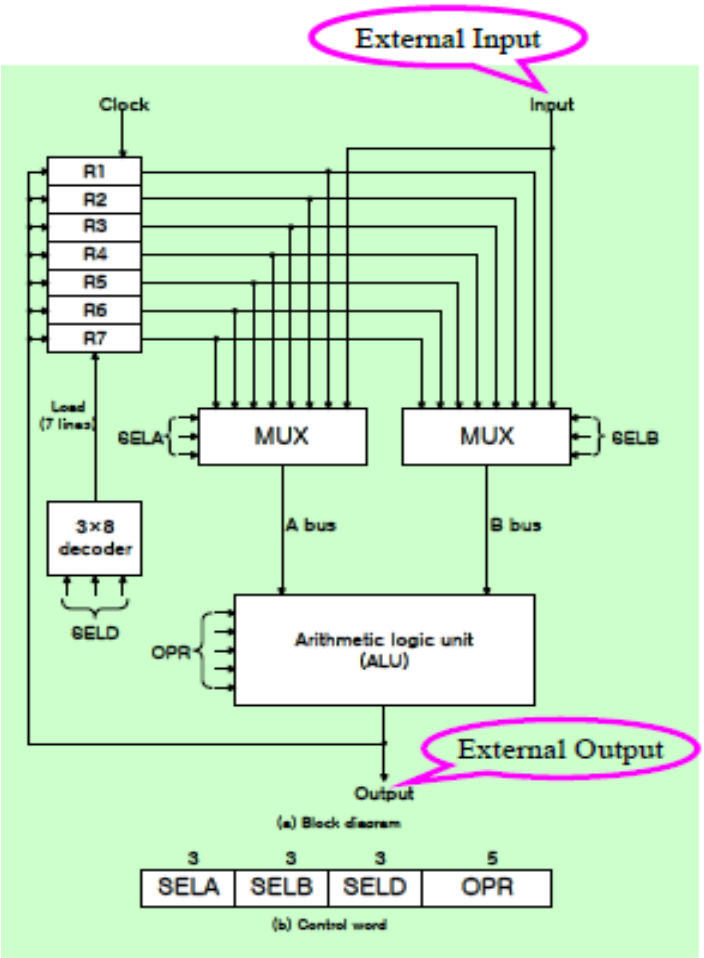
General Register Organization

Register: Memory locations are needed for storing *pointers, counters, return address, temporary results, and partial products*during multiplication .Memory access is the most time-consuming operation in a computer. *I*ts More convenient and efficient way is to store intermediate values in processor registers

Bus organization for 7 CPU registers :

-**2 MUX**: select one of 7 register or external data input by **SELA**and **SELB**
-**BUS A and BUS B**: form the inputs to a common ALU
-**ALU**: **OPR**determine the arithmetic or logic microoperation

»The result of the microoperation is available for external data output and also goes into the inputs of all the registers
-**3 X 8 Decoder**: select the register (by **SELD**) that receives the information from ALU



OPERATION OF CONTROL UNIT
The control unit
Directs the information flow through ALU by
- Selecting various *Components* in the system

- Selecting the *Function* of ALU

Example: R1 <- R2 + R3

[1] MUX A selector (SELA): BUS A <- R2

[2] MUX B selector (SELB): BUS B <- R3

[3] ALU operation selector (OPR): ALU to ADD

[4] Decoder destination selector (SELD): R1 <- Out Bus

Control Word

3	3	3	5
SELA	SELB	SELD	OPR

Encoding of register selection fields

Binary Code	SELA	SELB	SELD
000	Input	Input	None
001	R1	R1	R1
010	R2	R2	R2
011	R3	R3	R3
100	R4	R4	R4
101	R5	R5	R5
110	R6	R6	R6
111	R7	R7	R7

ALU CONTROL

Encoding of ALU operations

OPR Select	Operation	Symbol
00000	Transfer A	TSFA
00001	Increment A	INCA
00010	ADD A + B	ADD
00101	Subtract A - B	SUB
00110	Decrement A	DECA
01000	AND A and B	AND
01010	OR A and B	OR
01100	XOR A and B	XOR
01110	Complement A	COMA
10000	Shift right A	SHRA
11000	Shift left A	SHLA

Examples of ALU Microoperations

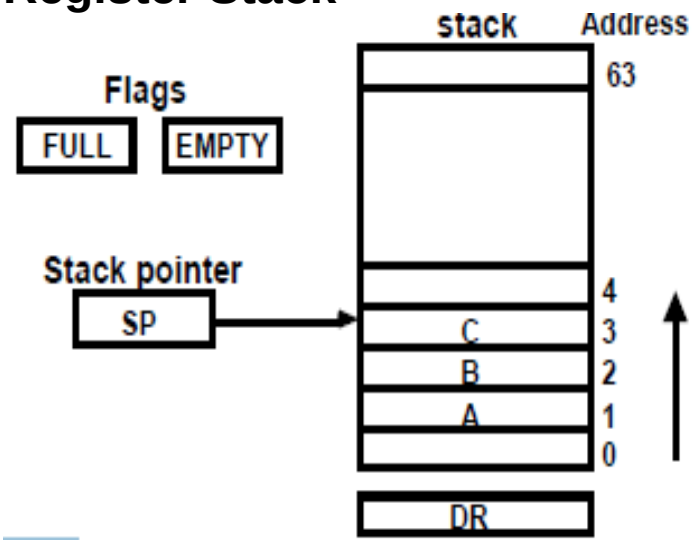
	Symbolic Designation				
Microoperation	SELA	SELB	SELD	OPR	Control Word
R1 ← R2 - R3	R2	R3	R1	SUB	010 011 001 00101
R4 ← R4 ∨ R5	R4	R5	R4	OR	100 101 100 01010
R6 ← R6 + 1	R6	-	R6	INCA	110 000 110 00001
R7 ← R1	R1	-	R7	TSFA	001 000 111 00000
Output ← R2	R2	-	None	TSFA	010 000 000 00000
Output ← Input	Input	-	None	TSFA	000 000 000 00000
R4 ← shl R4	R4	-	R4	SHLA	100 000 100 11000
R5 ← 0	R5	R5	R5	XOR	101 101 101 01100

REGISTER STACK ORGANIZATION

Stack

- Very useful feature for nested subroutines, nested loops control
- Also efficient for arithmetic expression evaluation
- Storage which can be accessed in LIFO
- Pointer: SP
- Only PUSH and POP operations are applicable

Register Stack



Push, Pop operations

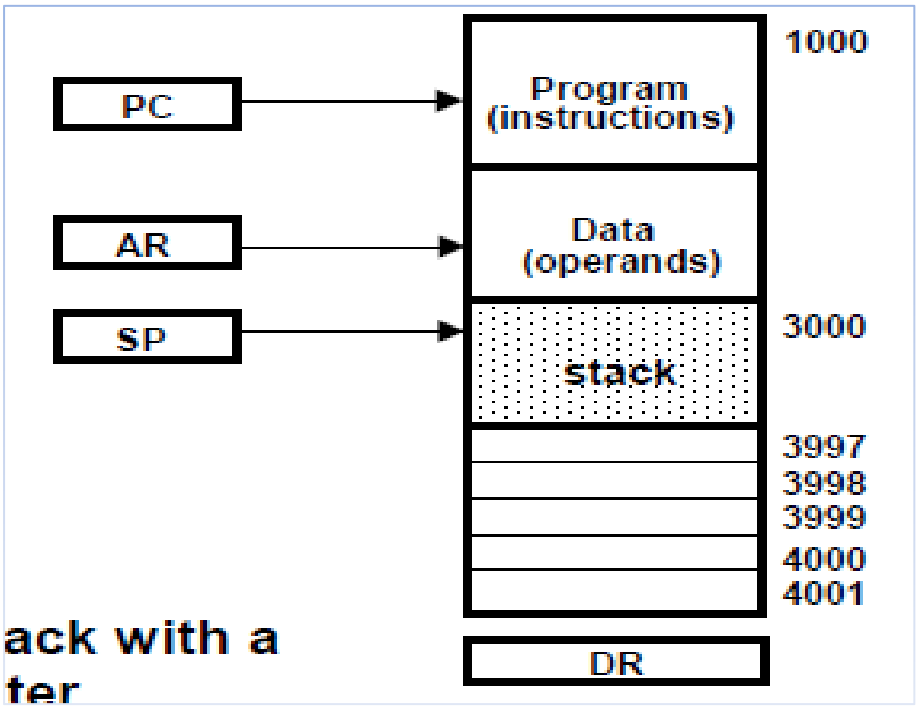
/* Initially, SP = 0, EMPTY = 1, FULL = 0 */

PUSH POP

SP ← SP + 1 DR ← M[SP]
M[SP]←DR SP ←SP - 1
If (SP = 0) then (FULL ← 1)If (SP = 0) then (EMPTY←1)
EMPTY ←0 FULL ← 0

MEMORY STACK ORGANIZATION

Memory with Program, Data,and Stack Segments

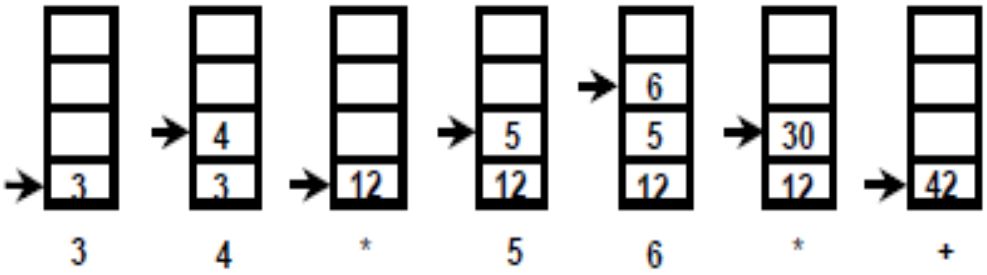


- A portion of memory is used as a stack with aprocessor register as a stack pointer
- PUSH: SP ← SP - 1
M[SP] ← DR
- POP: DR ← M[SP]
SP ←SP + 1
- Most computers do not provide hardware to checkstack overflow (full stack) or underflow(empty stack)

REVERSE POLISH NOTATION

Arithmetic Expressions: A + B
A + B Infix notation
+ A B Prefix or Polish notation
A B + Postfix or reverse Polish notation
- The reverse Polish notation is very suitable for stackmanipulation
Evaluation of Arithmetic Expressions . Any arithmetic expression can be expressed in parenthesis-freePolish notation, including reverse Polish notation

(3 * 4) + (5 * 6) ⇒ 3 4 * 5 6 * +



INSTRUCTION FORMAT

Instruction fields:
OP-code field - specifies the operation to be performed
Address field - designates memory address(es) or a processor register(s)
Mode field - specifies the way the operand or theeffective 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-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[A] */
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 */
- Computers with two-address instructions are most common

- One-Address Instructions

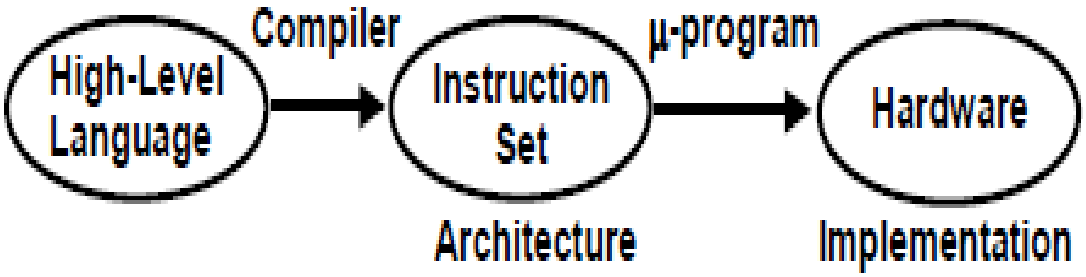
- Use an implied AC register for all data manipulation
- Program to evaluate X = (A + B) * (C + D) :
Instruction Format
LOAD A /* AC ← M[A] */
ADD B /* AC ← AC + M[B] */
STORE T /* M[T] ← AC */
LOAD C /* AC ← M[C] */
ADD D /* AC ← AC + M[D] */
MUL T /* AC ← AC * M[T] */
STORE X /* M[X] ← AC */

- 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 INSTRUCTION SET COMPUTERS

Historical Background: IBM System/360, 1964
- The real beginning of modern computer architecture
- Distinction between *Architecture* and *Implementation*
- Architecture: The abstract structure of a computer
seen by an assembly-language programmer



Continuing growth in semiconductor memory and microprogramming
-> A much richer and complicated instruction sets
=> CISC(Complex Instruction Set Computer)
- Arguments advanced at that time
Richer instruction sets would simplify compilers
Richer instruction sets would alleviate the software crisis
- move as much functions to the hardware as possible
- close *Semantic Gap* between machine language
and the high-level language
Richer instruction sets would improve *architecture quality*

ARCHITECTURE DESIGN PRINCIPLES

* Large microprograms would add little or nothingto the cost of the machine
<- Rapid growth of memory technology
-> Large General Purpose Instruction Set

* Microprogram is much faster than the machine instructions
<- Microprogram memory is much faster than main memory
-> Moving the software functions intoMicroprogram for the high performance machines

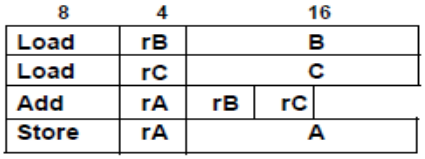
* Execution speed is proportional to the program size
-> Architectural techniques that led to small program
-> High performance instruction set

* Number of registers in CPU has limitations
-> Very costly
-> Difficult to utilize them efficiently

COMPARISONS OF EXECUTION MODELS

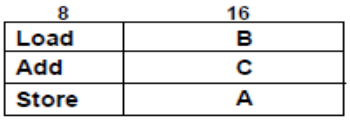
A <– B + C Data: 32-bit

Register-to-register



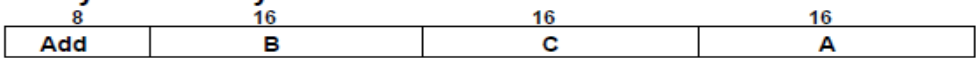
I = 104b; D = 96b; M = 200b

Memory-to-register



I = 72b; D = 96b; M = 168b

Memory-to-memory



I = 56b; D = 96b; M = 152b

Changes in the Implementation World in 70’s

* Main Memory is no longer 10 times slower thanMicroprogram memory
-> microprogram rather slows down the speed

* Caches had been invented
-> Further improvement on the Main Memory speed

* Compilers were subsetting architectures

CRITICIS ON COMPLEX INSTRUCTION SET COMPUTERS

Complex Instruction Set Computers - CISC

High Performance General Purpose Instructions

- Complex Instruction
-> Format, Length, Addressing Modes
-> Complicated instruction cycle control due to the complexdecoding HW and decoding process

- Multiple memory cycle instructions
-> Operations on memory data
-> Multiple memory accesses/instruction

- Microprogrammed control is necessity
-> Microprogram control storage takessubstantial portion of CPU chip area
-> Semantic Gap is large between machineinstruction and microinstruction

- General purpose instruction set includes all the features required by individually different applications
-> When any one application is running, all the featuresrequired by the other applications are extra burden tothe application

PHYLOSOPHY OF RISC

Reduce the semantic gap betweenmachine instruction and microinstruction

1-Cycle instruction

Most of the instructions complete their execution in 1 CPU clock cycle - like a microoperation

* Functions of the instruction (على النقيض من ذلكcontrast to CISC)
- Very simple functions
- Very simple instruction format
- Similar to microinstructions
=> No need for microprogrammed control
* Register-Register Instructions
- Avoid memory reference instructions exceptLoad and Store instructions
- Most of the operands can be found in the registers instead of main memory
=> Shorter instructions

=> Uniform instruction cycle
=> Requirement of large number of registers
* Employ instruction pipeline

ARCHITECTURAL METRIC

A <– B + C
B <– A + C
D <– D - B

Register-to-register (Reuse of Operands)

8	4	16
Load	rB	B
Load	rC	C
Add	rA	rB rC
Store	rA	A
Add	rB	rA rC
Store	rB	B
Load	rD	D
Sub	rD	rD rB
Store	rD	D

I = 228b
D = 192b
M = 420b

Register-to-register (Compiler allocates Operands in registers)

8	4	4	4
Add	rA	rB	rC
Add	rB	rA	rC
Sub	rD	rD	rB

I = 60b
D = 0b
M = 60b

Memory-to-memory

8	16	16	16
Add	B	C	A
Add	A	C	B
Sub	B	D	D

I = 168b
D = 288b
M = 456b

Common RISC Characteristics

- Operations are register-to-register, with only LOAD andSTORE accessing memory
- The operations and addressing modes are reduced

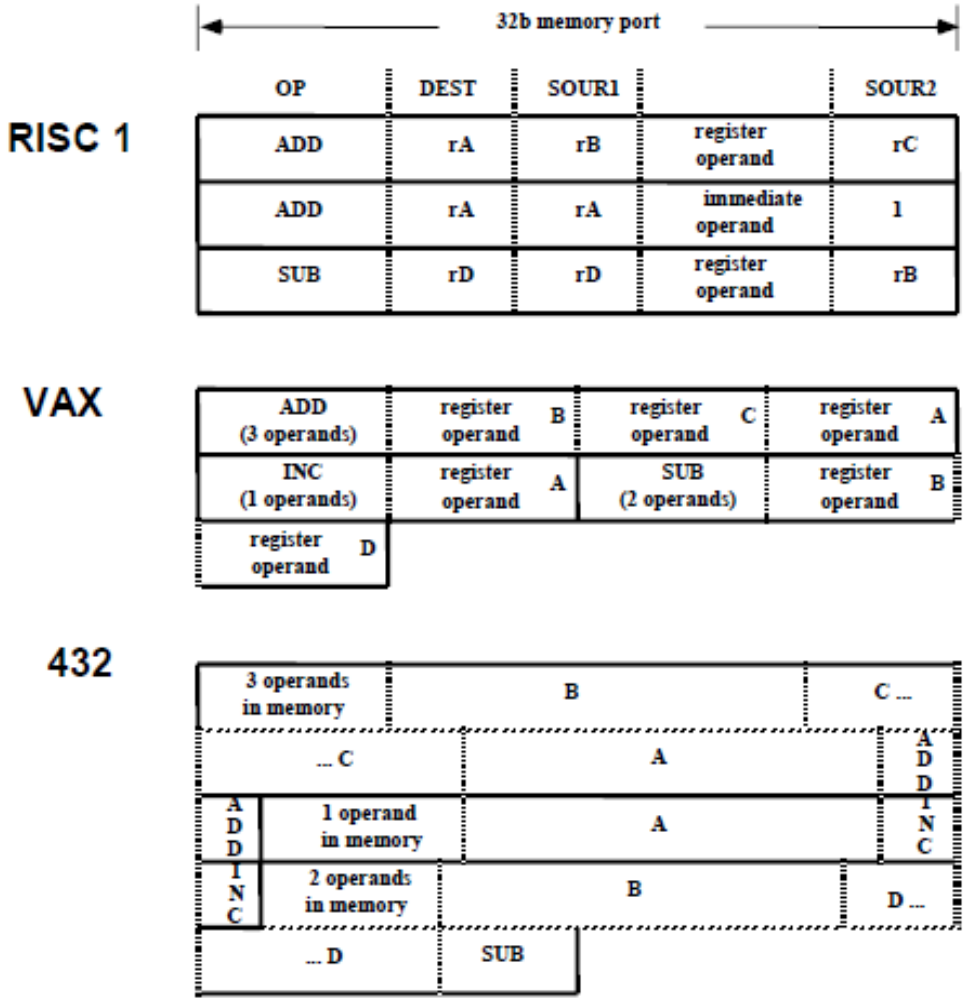
Instruction formats are simple and do not crossword boundaries

- RISC branches avoid pipeline penalties - delayed branch.

Characteristics of Initial RISC Machines

	IBM 801	RISC I	MIPS
Year	1980	1982	1983
Number of instructions	120	39	55
Control memory size	0	0	0
Instruction size (bits)	32	32	32
Technology	ECL MSI	NMOS VLSI	NMOS VLSI
Execution model	reg-reg	reg-reg	reg-reg

COMPARISON OF INSTRUCTION SEQUENCE



-Relatively few instructions
- Relatively few addressing modes
- Memory access limited to load and store instructions
- All operations done within the registers of the CPU
- Fixed-length, easily decoded instruction format
- Single-cycle instruction format
- Hardwired rather than microprogrammed control

Advantages of RISC
- VLSI Realization
- Computing Speed
- Design Costs and Reliability
- High Level Language Support

Pipeline And Vector Processing

Parallel Processing

Execution of Concurrent Events in the computing process to achieve faster Computational Speed

The purpose of parallel processing is to speed up the computer processing capability and increase its throughput, that is, the amount of processing that can be accomplished during a given interval of time.

The amount of hardware increases with parallel processing, and with it, the cost of the system increases.

However, technological developments have reduced hardware costs to the point where parallel processing techniques are economically feasible.

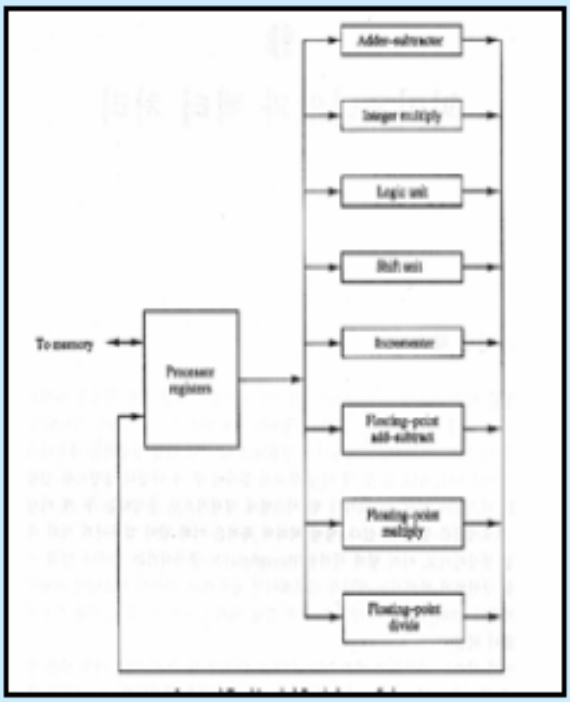
Parallel processing according to levels of complexity

At the lower level

Serial Shift register VS parallel load registers

At the higher level

Multiplicity of functional units that performer identical or different operations simultaneously.



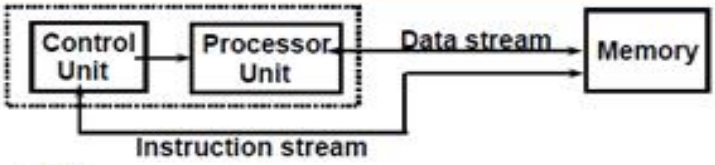
Parallel Computers

Architectural Classification

- Flynn's classification
  - » Based on the multiplicity of *Instruction Streams* and *Data Streams*
  - » Instruction Stream
    - Sequence of Instructions read from memory
  - » Data Stream
    - Operations performed on the data in the processor

		Number of *Data Streams*	
		Single	Multiple
Number of *Instruction Streams*	Single	SISD	SIMD
	Multiple	MISD	MIMD

SISD COMPUTER SYSTEMS



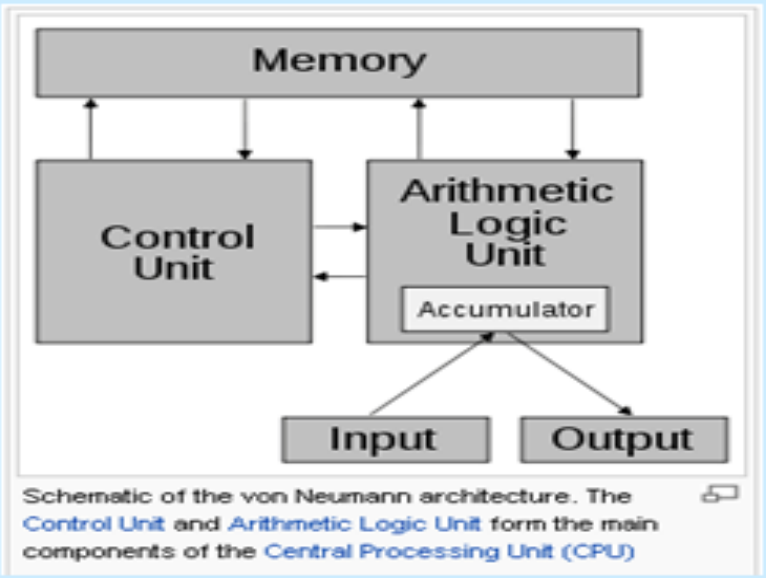
Characteristics
- Standard von Neumann machine
- Instructions and data are stored in memory
- One operation at a time

Limitations

Von Neumann bottleneck

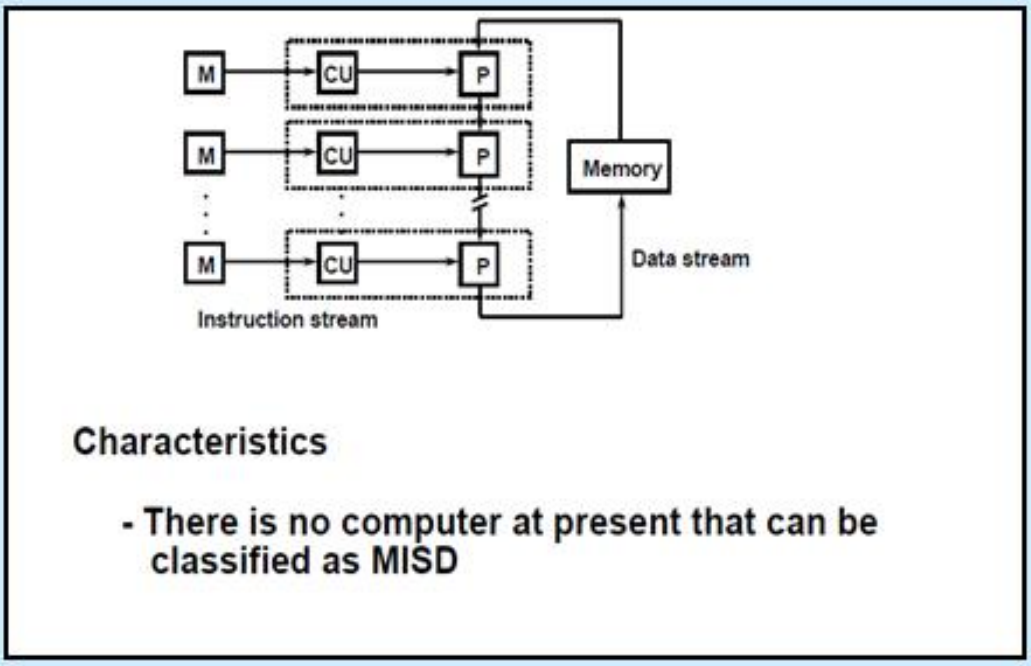
Maximum speed of the system is limited by the *Memory Bandwidth* (bits/sec or bytes/sec)
- Limitation on *Memory Bandwidth*
- Memory is shared by CPU and I/O

Von Neumann Architecture



Schematic of the von Neumann architecture. The Control Unit and Arithmetic Logic Unit form the main components of the Central Processing Unit (CPU)

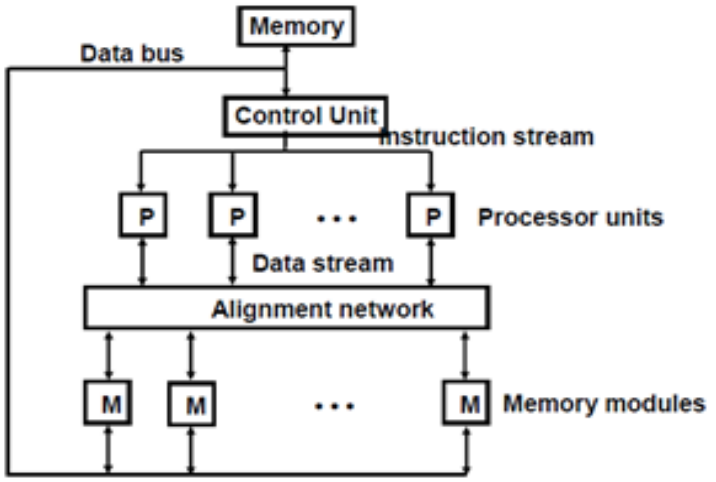
MISD COMPUTER SYSTEMS



Characteristics

- There is no computer at present that can be classified as MISD

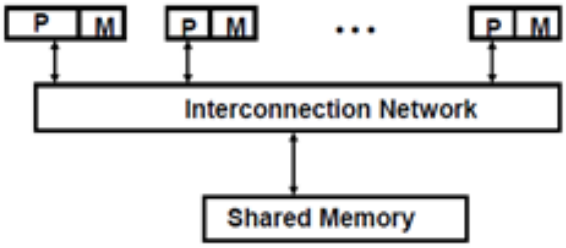
SIMD COMPUTER SYSTEMS



Characteristics

- Only one copy of the program exists
- A single controller executes one instruction at a time

MIMD COMPUTER SYSTEMS



Characteristics

- Multiple processing units
- Execution of multiple instructions on multiple data

Types of MIMD computer systems

- Shared memory multiprocessors
- Message-passing multicomputers

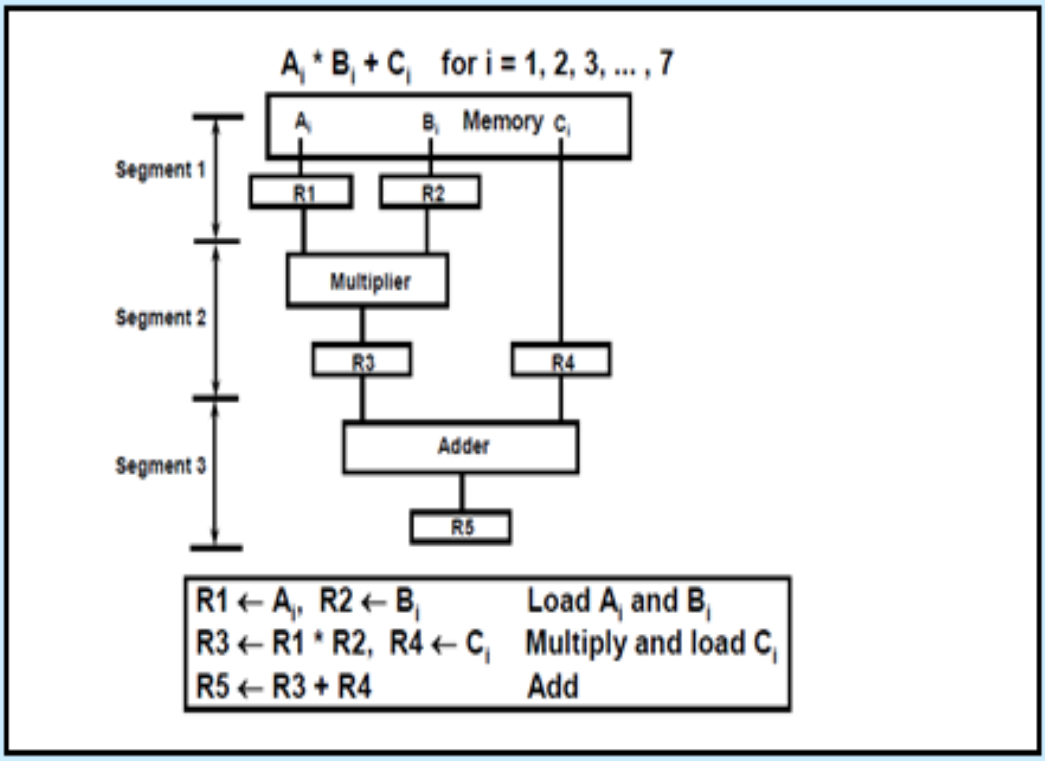
PIPELINING

A technique of decomposing a sequential process into suboperations, with each subprocess being executed in a partial dedicated segment that operates concurrently with all other segments.

A pipeline can be visualized as a collection of processing segments through which binary information flows.

The name "pipeline" implies a flow of information analogous to an industrial assembly line.

Example of the Pipeline Organization

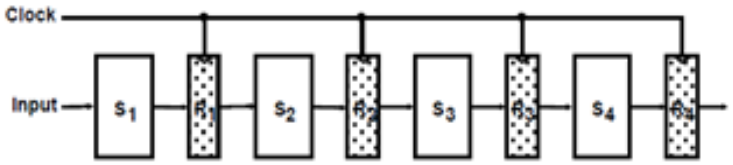


OPERATIONS IN EACH PIPELINE STAGE

Clock Pulse Number	Segment 1 R1	Segment 1 R2	Segment 2 R3	Segment 2 R4	Segment 3 R5
1	A1	B1			
2	A2	B2	A1 * B1	C1	
3	A3	B3	A2 * B2	C2	A1 * B1 + C1
4	A4	B4	A3 * B3	C3	A2 * B2 + C2
5	A5	B5	A4 * B4	C4	A3 * B3 + C3
6	A6	B6	A5 * B5	C5	A4 * B4 + C4
7	A7	B7	A6 * B6	C6	A5 * B5 + C5
8			A7 * B7	C7	A6 * B6 + C6
9					A7 * B7 + C7

GENERAL PIPELINE

General Structure of a 4-Segment Pipeline



Space-Time Diagram

Segment	1	2	3	4	5	6	7	8	9
1	T1	T2	T3	T4	T5	T6			
2		T1	T2	T3	T4	T5	T6		
3			T1	T2	T3	T4	T5	T6	
4				T1	T2	T3	T4	T5	T6

Clock cycles

Behavior of the pipeline is illustrated with a space time diagram.
Space time diagram:
This shows the segment utilization as a function of time.

Space Time diagram:

- The horizontal axis displays the time in clock cycle and vertical axis gives the segment number
- Diagram shows 6 task (T1 to T6)executed in four segment

Task :
is defined as the total operation performed going through all the segment in the pipeline

Speedup ratio of pipeline

Consider

- **k: segment pipeline with clock cycle time t_p to execute n tasks**
- **first task T1 requires a time equal kt_p to complete its operation since there are k segments in the pipe .**
- **Remaining n-1 tasks emerge from the pipe at the rate of one task per clock cycle and they will complete after a time equal to (n-1)tp.**
- **Therefore to complete n task using k-segement pipeline requires**
  K+(n-1) clock cycle
- **Example 4 segment , 6task**
  time required to complete op. 4+(6-1)=9
  clock cycle

Cont.

- For nonpipeline unit that perform the same operation and takes a time equal to t_n to complete each task.
- The total time required for n tasks $=nt_n$
- Speedup of a pipeline processing over an equivalent nonpipeline processing is defined by the ratio
- $S=nt_n / (K+n-1)t_p$
- As the number of tasks increases , n beomes larger the k-1, and k+n-1 approaches the value of n under this condition ,the speedup becomes $S=t_n / t_p$
- If we assume that the time it takes to process a task is the same in the pipeline and nonpipeline circuit, tn=ktp
- Including the assumption speedup reduces to $S=Kt_p/t_p=K$
- This shows that the theoretical max. speedup that a pipeline can provide is k, where k is the no. of segment in the pipeline

INSTRUCTION CYCLE

Six Phases* in an Instruction Cycle

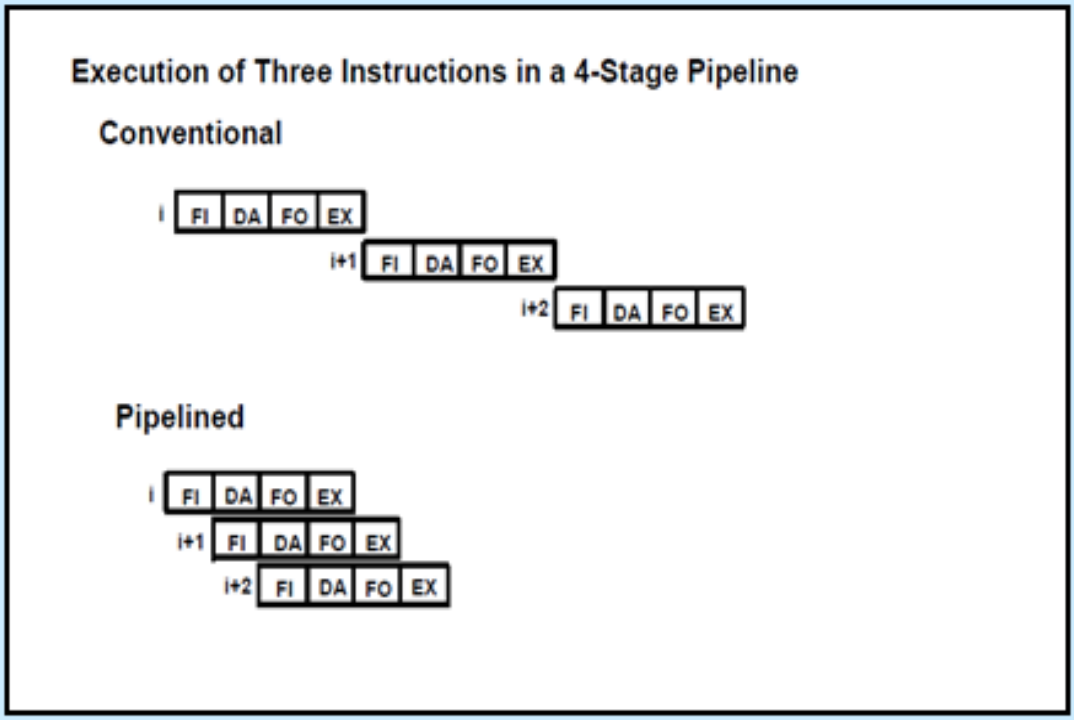
[1] Fetch an instruction from memory
[2] Decode the instruction
[3] Calculate the effective address of the operand
[4] Fetch the operands from memory
[5] Execute the operation
[6] Store the result in the proper place

* Some instructions skip some phases
* Effective address calculation can be done in the part of the decoding phase
* Storage of the operation result into a register is done automatically in the execution phase

==> 4-Stage Pipeline

[1] FI: Fetch an instruction from memory
[2] DA: Decode the instruction and calculate the effective address of the operand
[3] FO: Fetch the operand
[4] EX: Execute the operation

INSTRUCTION PIPELINE



VECTOR PROCESSING

There is a class of computational problems that are beyond the capabilities of conventional computer.

These problems are characterized by the fact that they require a vast number of computations that will take a conventional computer days or even weeks to complete.

VECTOR PROCESSING

Vector Processing Applications

- Problems that can be efficiently formulated in terms of vectors
  - Long-range weather forecasting
  - Petroleum explorations
  - Seismic data analysis
  - Medical diagnosis
  - Aerodynamics and space flight simulations
  - Artificial intelligence and expert systems
  - Mapping the human genome
  - Image processing

Vector Processor (computer)

Ability to process vectors, and related data structures such as matrices and multi-dimensional arrays, much faster than conventional computers

Vector Processors may also be pipelined

VECTOR PROGRAMMING

```
      DO  20  I = 1, 100
20    C(I) = B(I) + A(I)
```

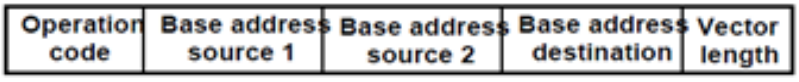
Conventional computer

```
      Initialize I = 0
20    Read A(I)
      Read B(I)
      Store C(I) = A(I) + B(I)
      Increment I = i + 1
      If I ≤ 100 goto 20
```

Vector computer

```
      C(1:100) = A(1:100) + B(1:100)
```

Vector Instruction Format



VECTOR INSTRUCTIONS

f1: V • V
f2: V • S
f3: V x V • V
f4: V x S • V

V: Vector operand
S: Scalar operand

Type	Mnemonic	Description (I = 1, ..., n)	
f1	VSQR	Vector square root	B(I) • SQR(A(I))
	VSIN	Vector sine	B(I) • sin(A(I))
	VCOM	Vector complement	A(I) • $\overline{A(I)}$
f2	VSUM	Vector summation	S • Σ A(I)
	VMAX	Vector maximum	S • max{A(I)}
f3	VADD	Vector add	C(I) • A(I) + B(I)
	VMPY	Vector multiply	C(I) • A(I) * B(I)
	VAND	Vector AND	C(I) • A(I) . B(I)
	VLAR	Vector larger	C(I) • max(A(I),B(I))
	VTGE	Vector test >	C(I) • 0 if A(I) < B(I)
			C(I) • 1 if A(I) > B(I)
f4	SADD	Vector-scalar add	B(I) • S + A(I)
	SDIV	Vector-scalar divide	B(I) • A(I) / S