Computer Architecture

CPU Architecture, Instruction Cycles, Formats, Sets

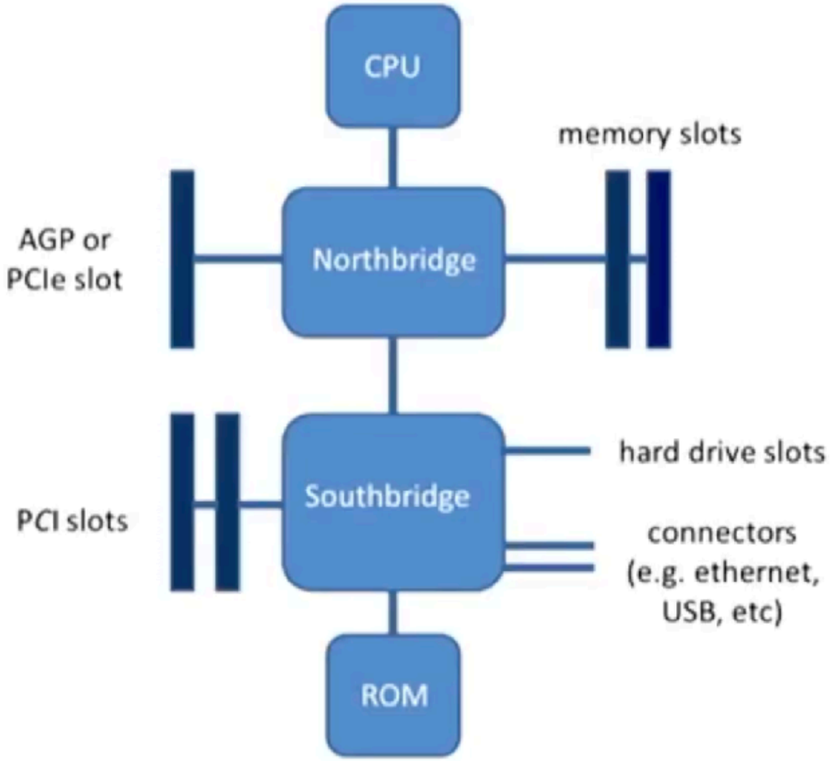
Dr. K. R. Kaza, IIIT Allahabad

Some publicly available teaching material from Prof. Chester Ribeiro, Dr. John Jose and others has been utilized.

Agenda

- Stored Program Concept
- Architecture Models - Von-Neumann/Harvard
- Basic Register Organization
- Instruction Execution Cycle
- Addressing Modes

What's inside a computer?



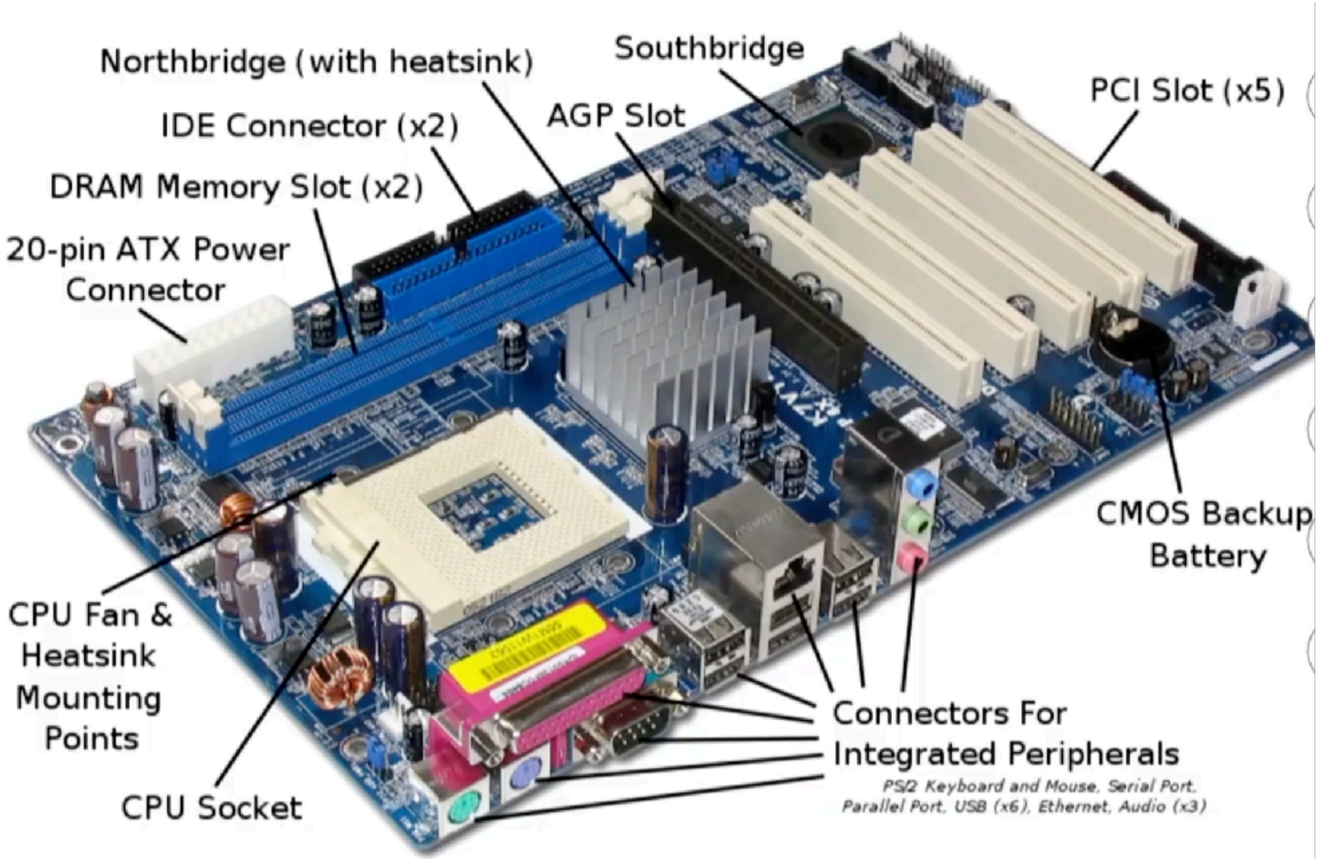
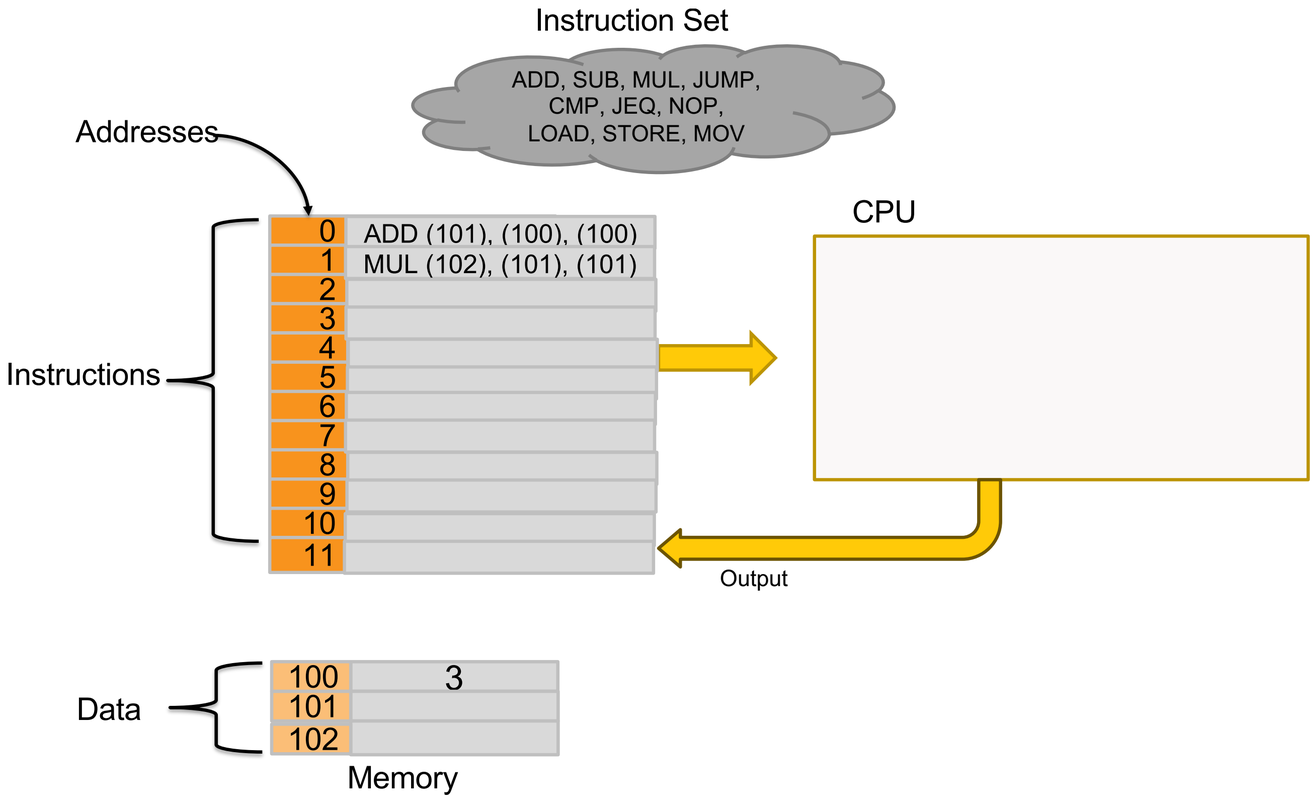


Image source: Prof. John Jose

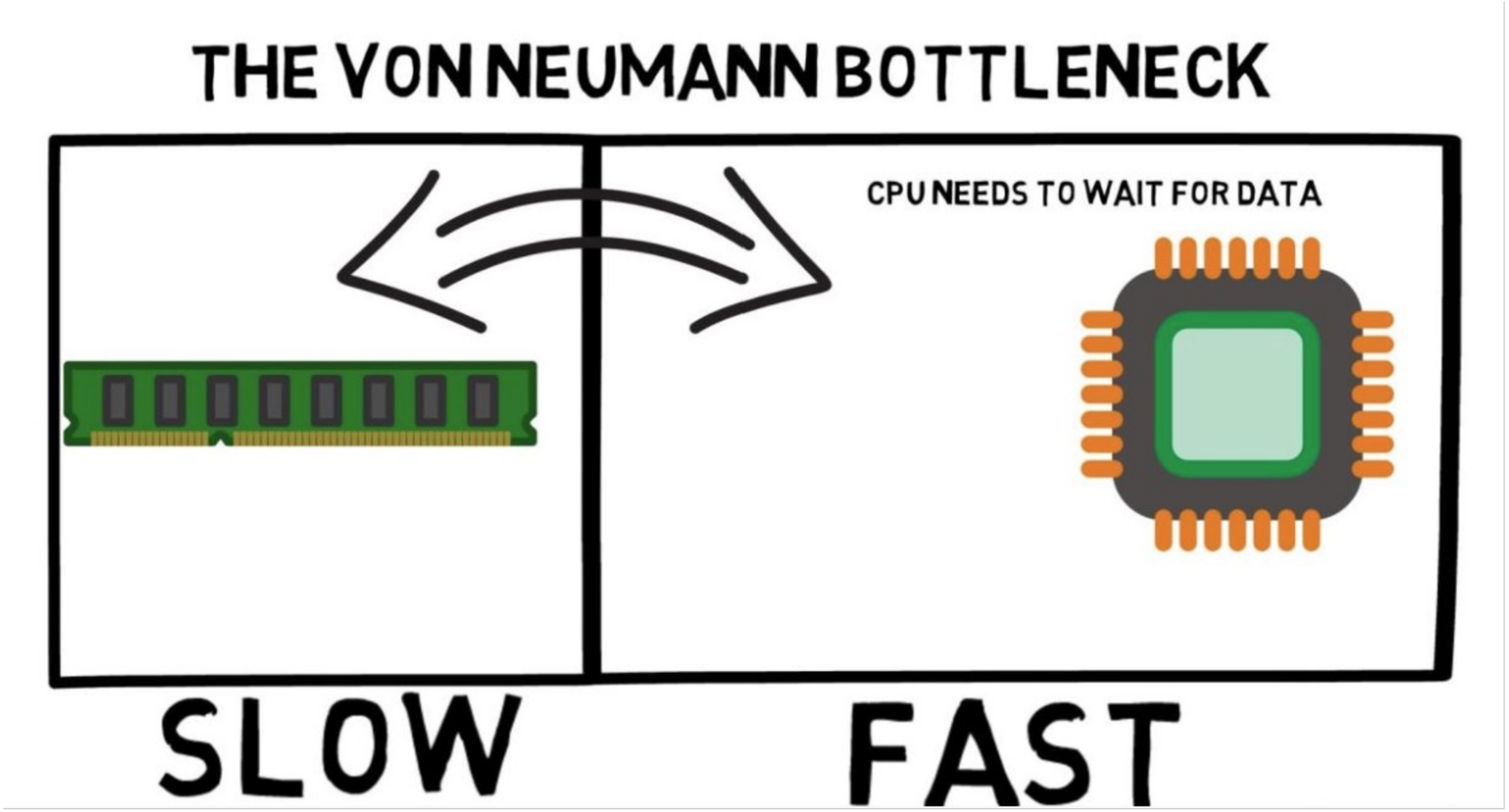
Stored Program Concept

- Von Neumann Architecture
- Program - written using a set of fixed instructions
- CPU fetches instructions from the programs in the memory
- Operates on data which is also stored in the memory
- Executed sequentially
- Programs can be modified like data



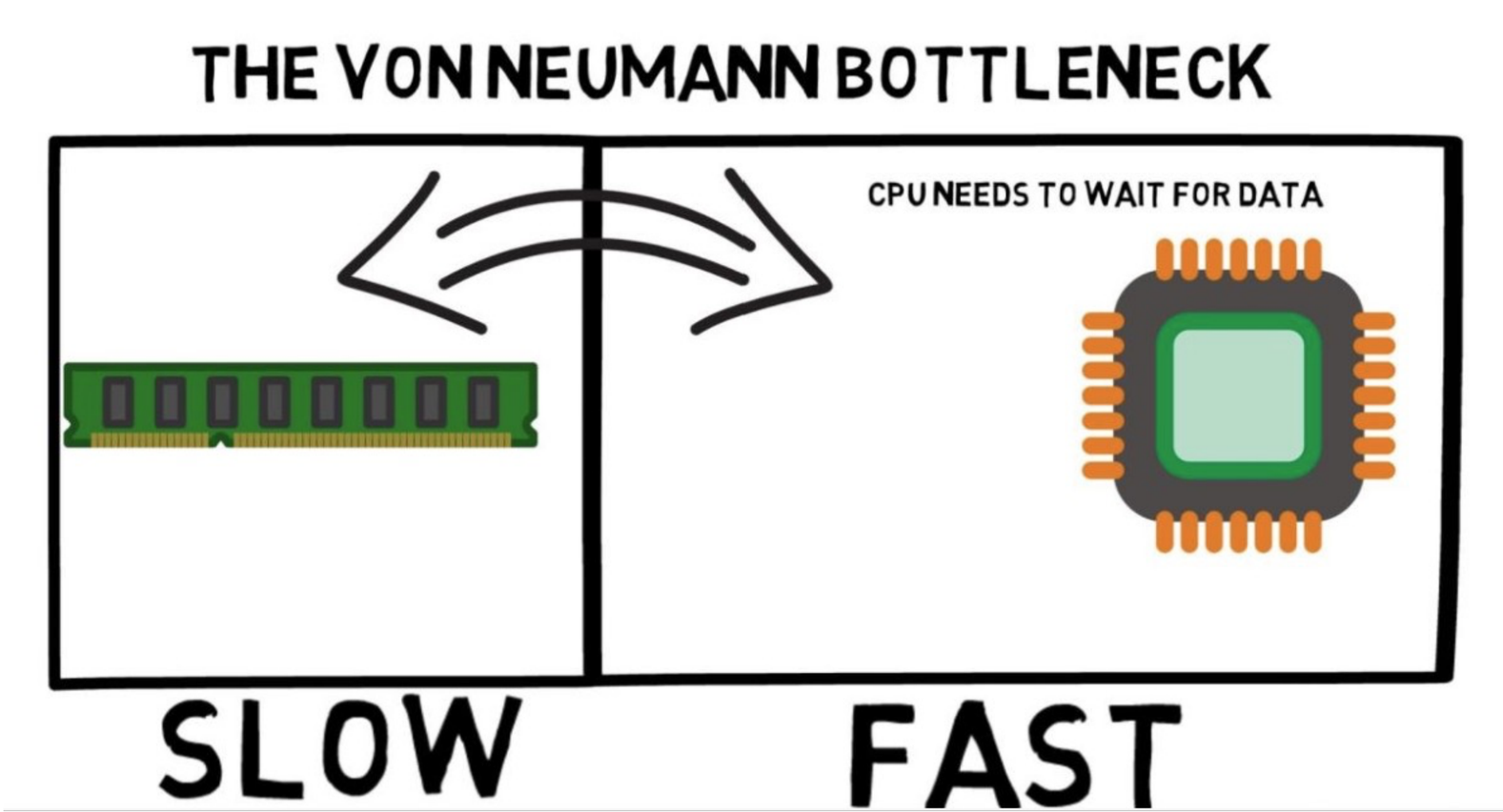
Von-Neumann Bottleneck

- Has to wait for data from the memory
- Gap between performance of CPU vs DRAM has widened over the years
- CPU is much faster than the memory

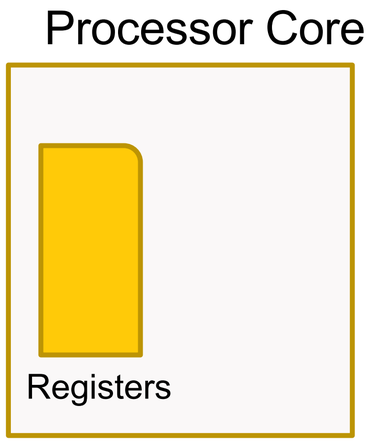


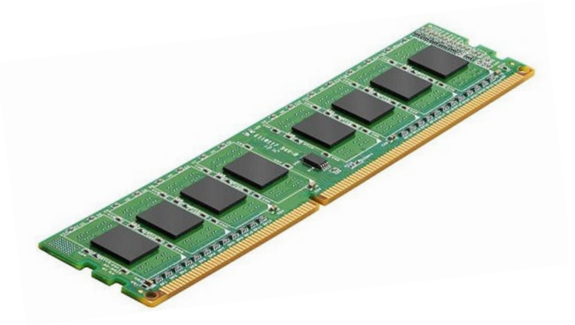
Registers

- Introduce registers which are much faster and closer to the CPU
- **But, very limited in number**
- **Typically around 16, 32, or 64 registers**

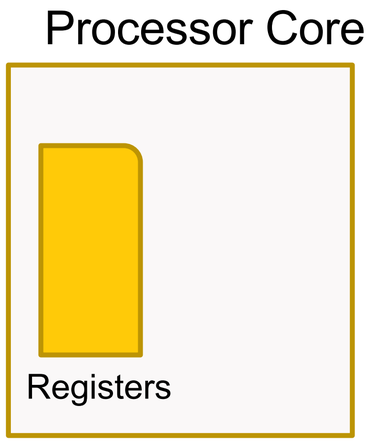


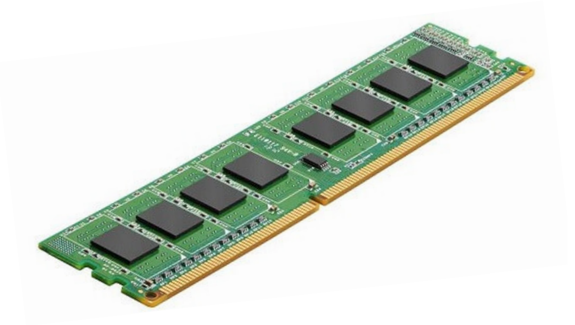
Registers vs RAM





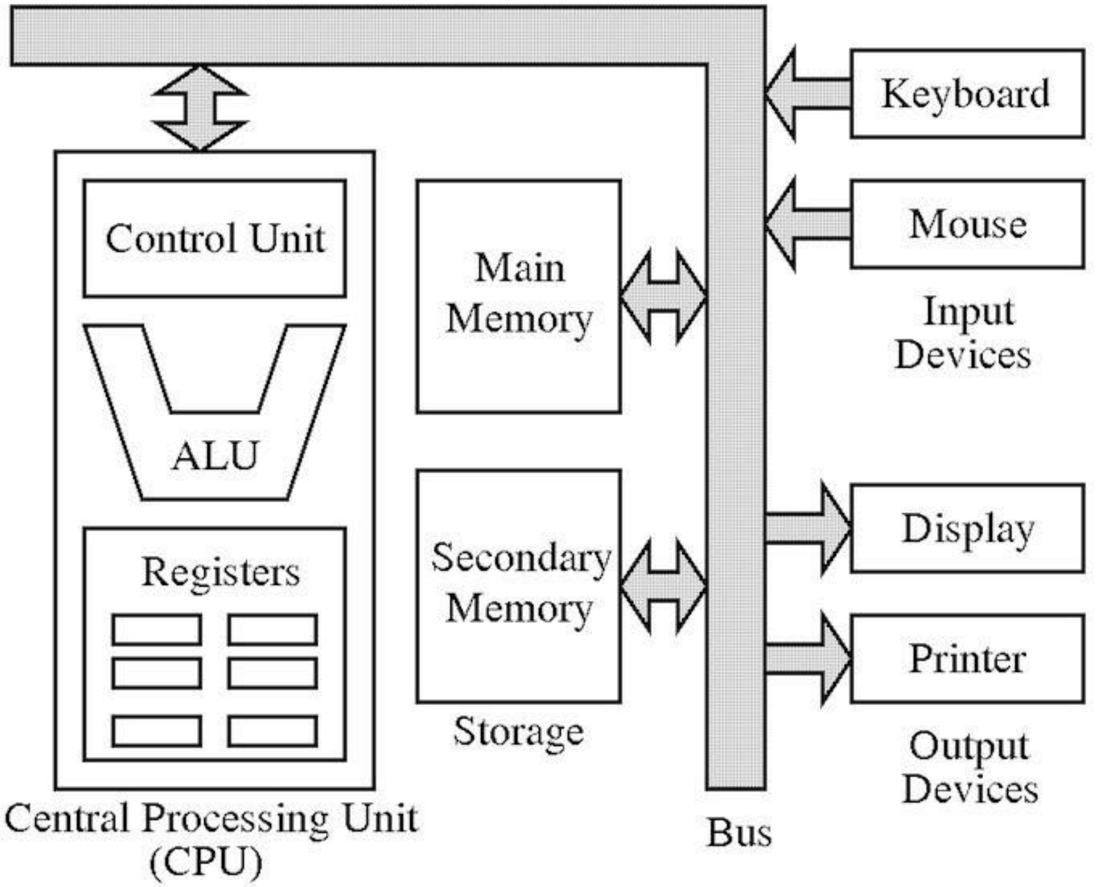
Program Transformations





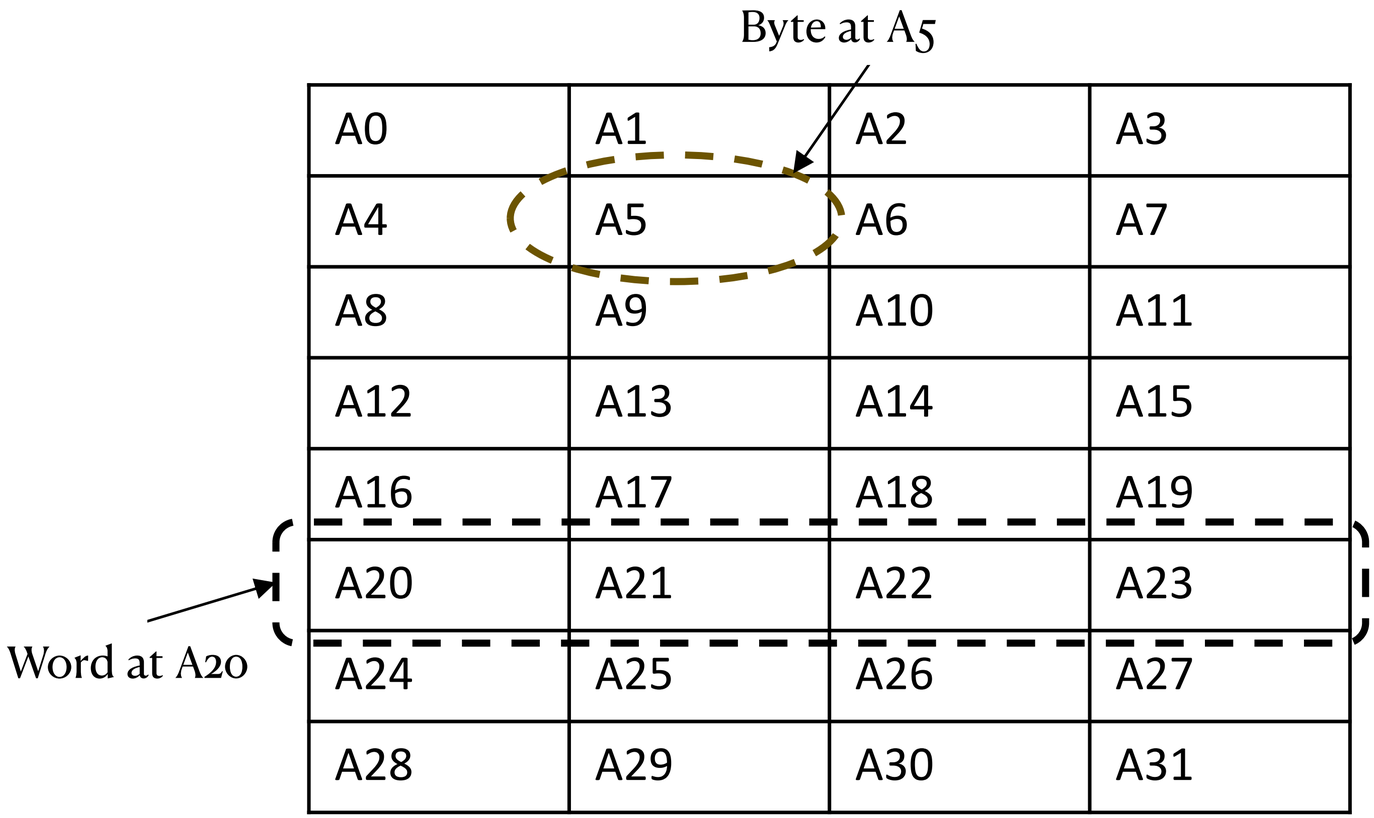
Stored Program Architecture

- Programs are usually stored in the secondary storage
- Loaded into the main memory
- CPU has direct access to main memory
- Secondary Storage > Main Memory larger > CPU Internal memory (registers)



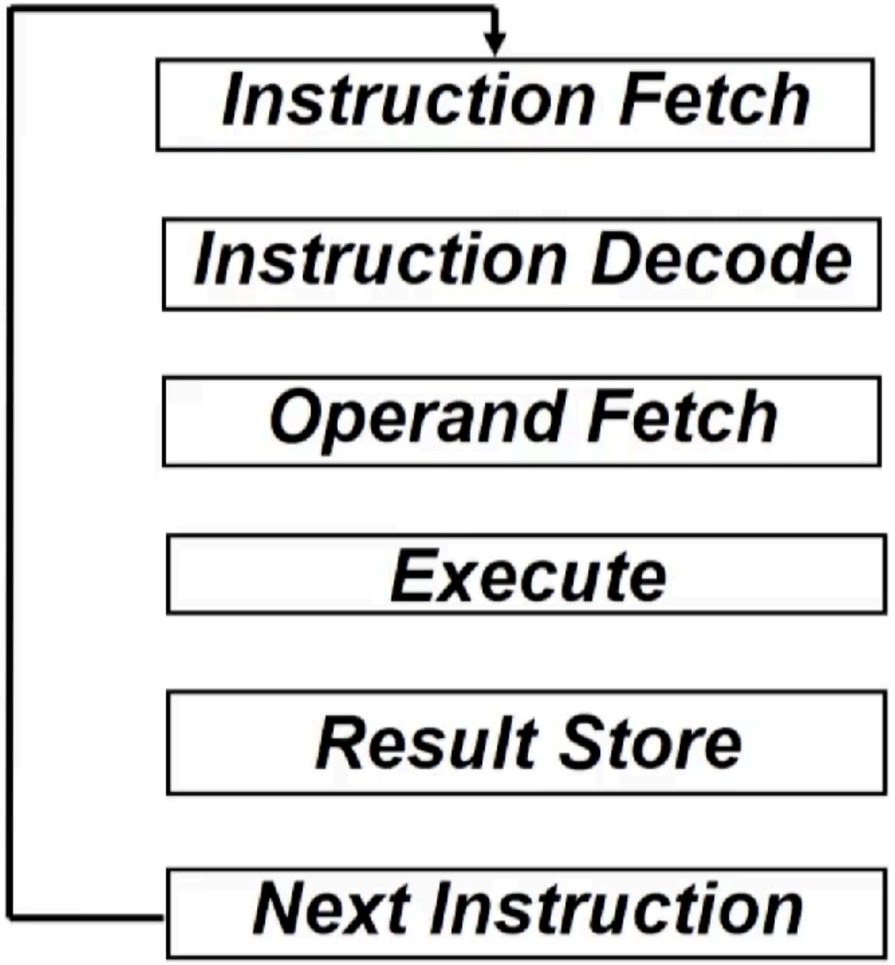
Main Memory (RAM)

- Much larger than CPU internal memory
- Can vary from a few MB to TB
- Made of transisters and capacitors
- Byte locations accessed with addresses
- Each word is consists of 4 bytes



Basic Instruction Execution Cycle

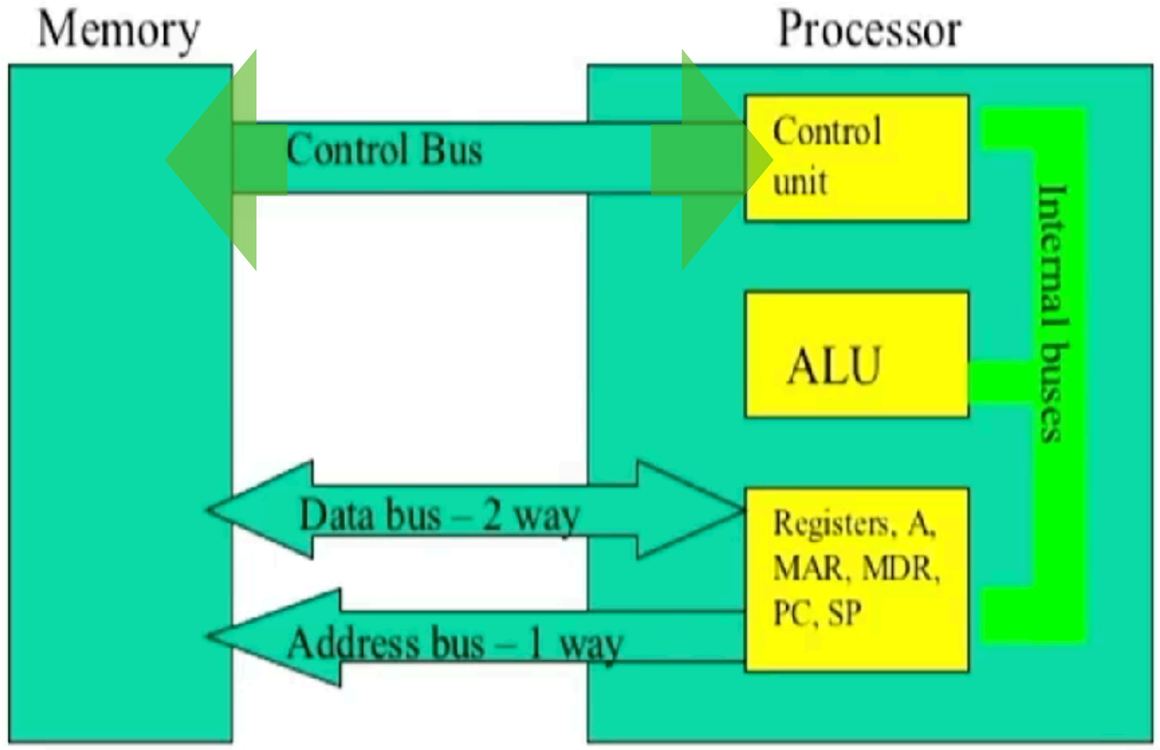
- Instruction Fetch - *From Memory* through the bus *to the Processor*
- Instruction Decode: Understanding the required actions and size
- Locate the operands and bring them
- Compute the operation
- Store the result in the memory or register
- Move to the next instruction



CPU-Memory Interaction

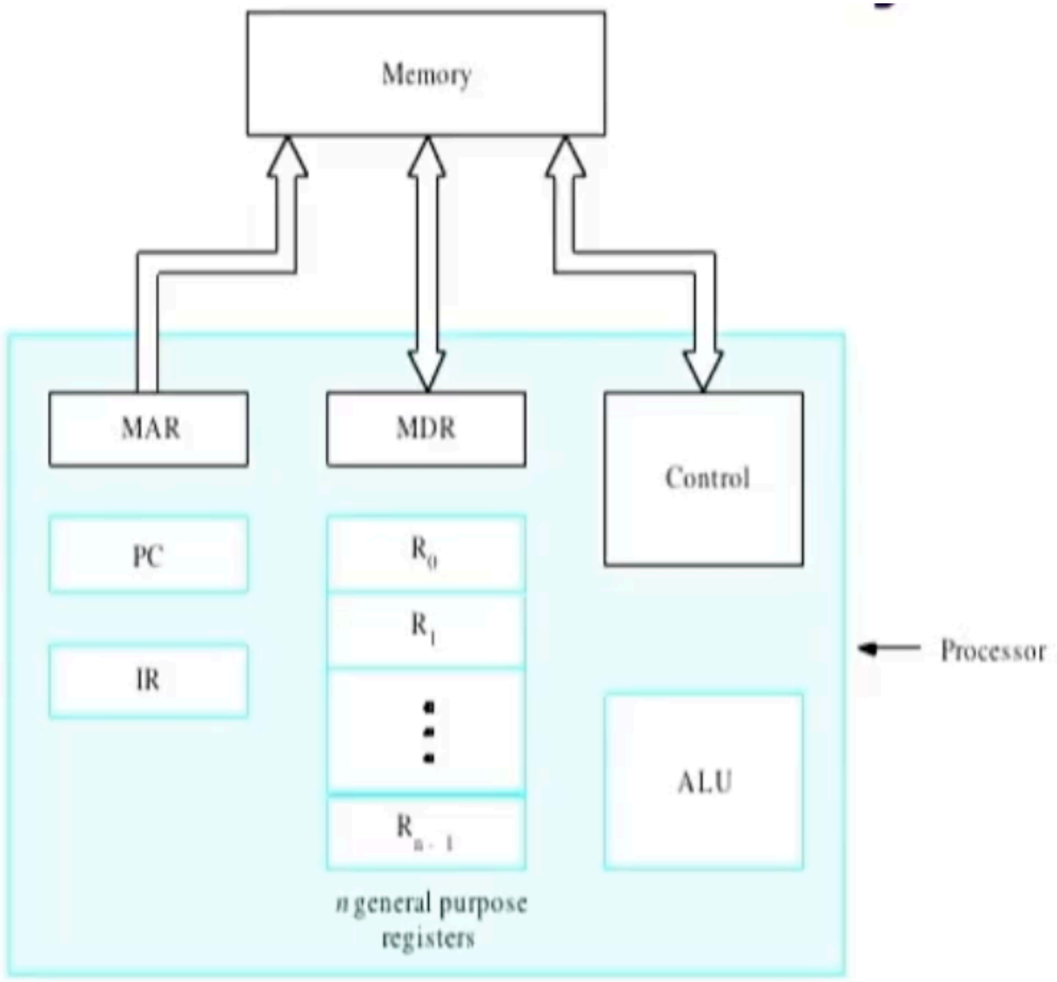
In an instruction cycle

- Through special wires - called bus
- Different buses are available - bidirectional and unidirectional
- CPU internal buses
- Address bus - specifies memory location
- Data bus - instruction/data fetch or store results of operation
- Control bus - Handshake/control signals for smooth communication.



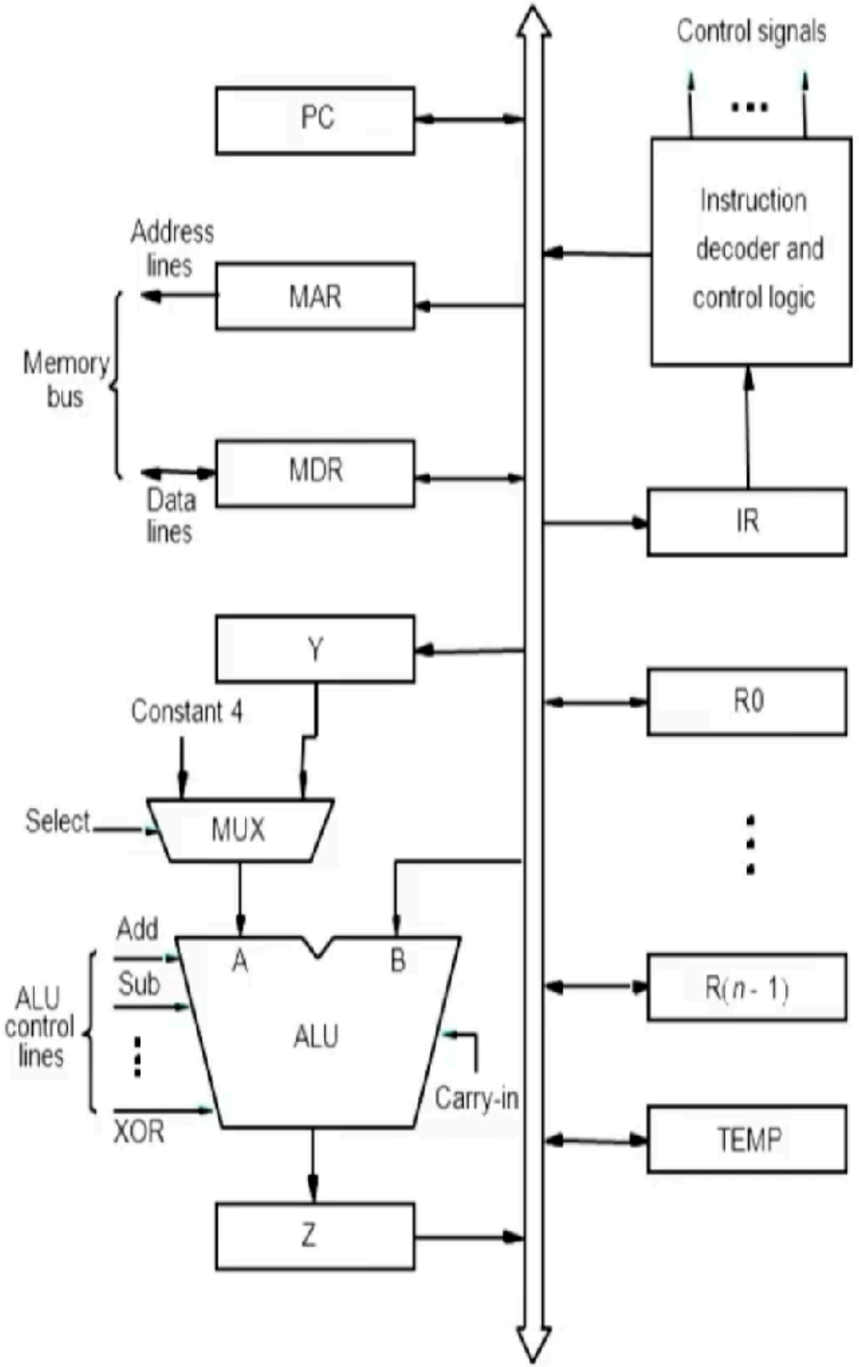
CPU - Register Organization

- General purpose registers - temporarily store data or values of operands
- MAR - memory address register acts a interface between CPU and memory
- MDR - memory data registrer
- Program Counter - address of the next instruction to be fetched and executed
- Instruction Register - Performs the decoding operation



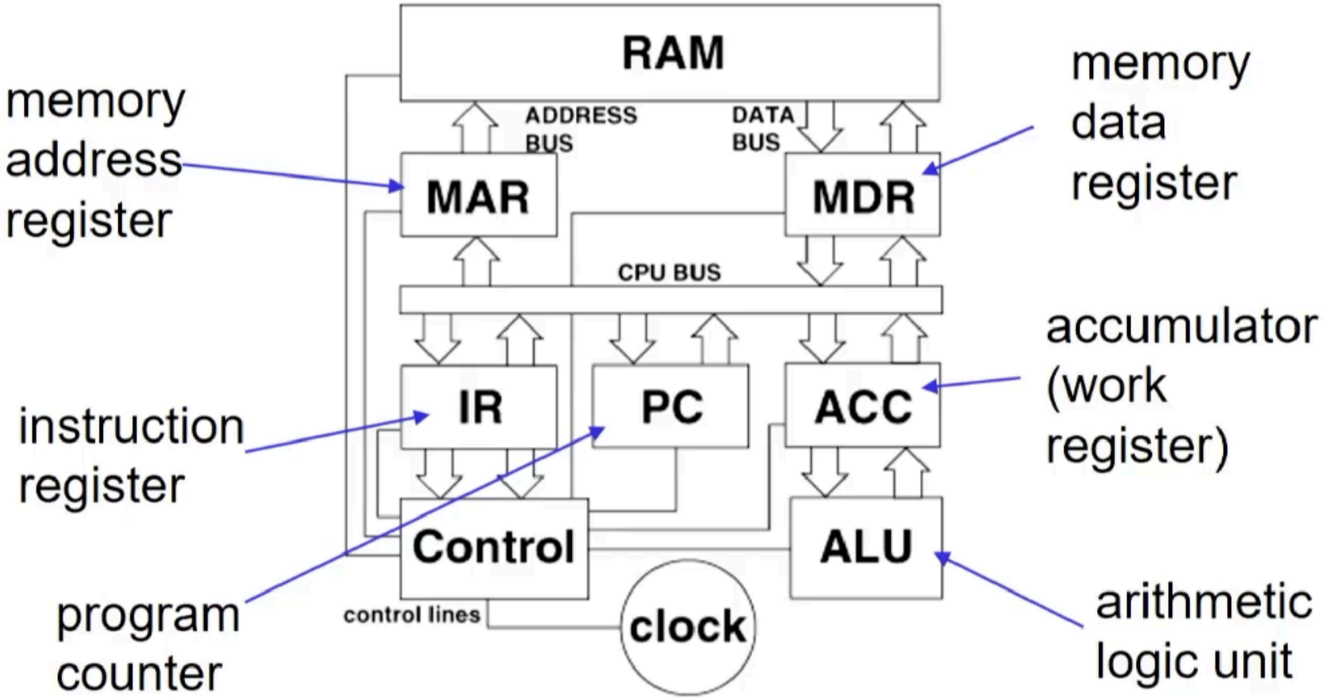
CPU - Register Organization

- General purpose registers temporarily store data - Ro ... R(n-1)
- Control unit gives instructions to ALU on the type of operation
- Two inputs to ALU - one from the bus and another from an internal register
- Program Counter
- Instruction Register
- MAR - memory address register
- MDR - memory data registrer



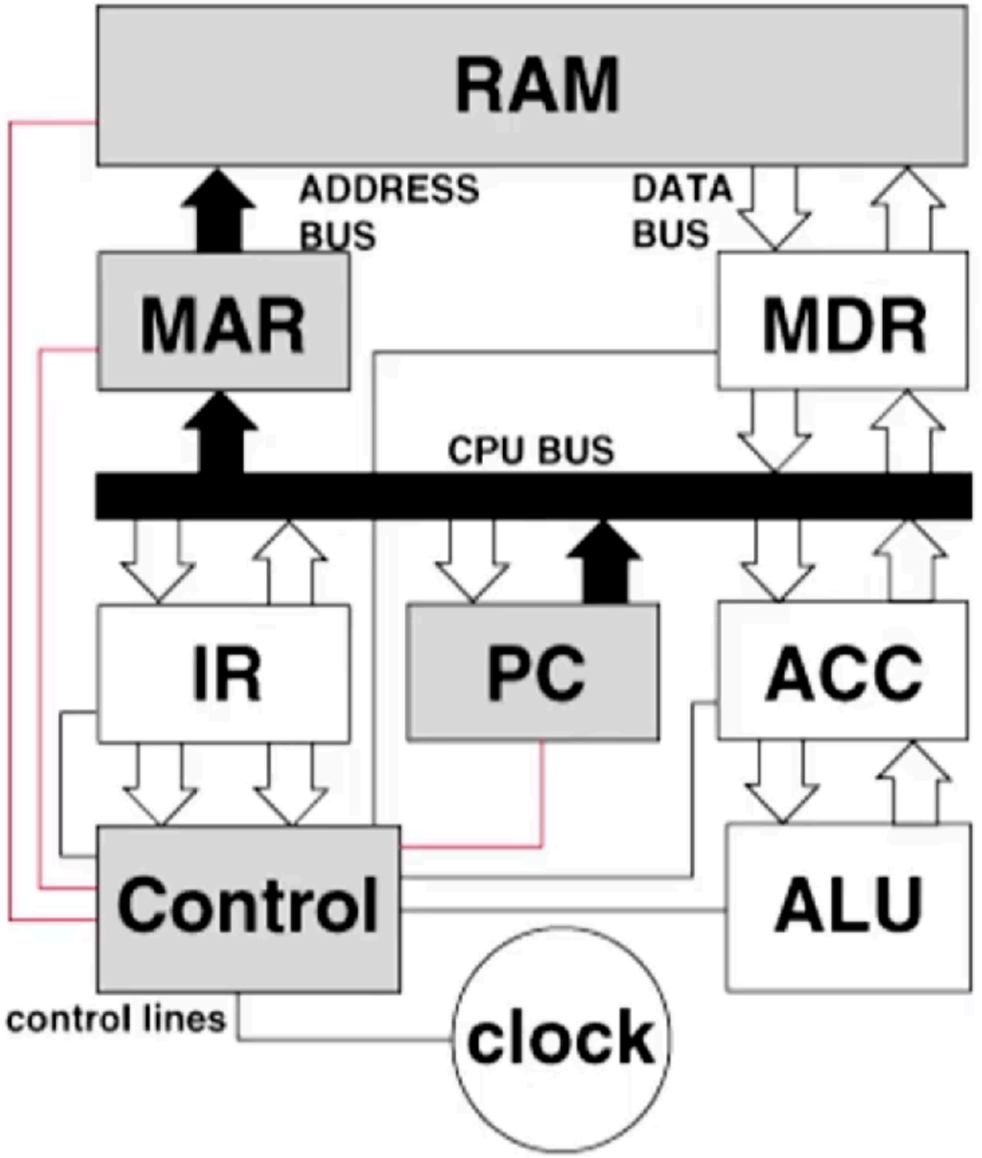
CPU Architecture

- MAR and MDR directly interface with the main memory
- The address of any memory location that needs to be accessed should be specified by MAR
- Any data which has been read or has to be written is specified through MDR
- Accumulator - Output of ALU operations



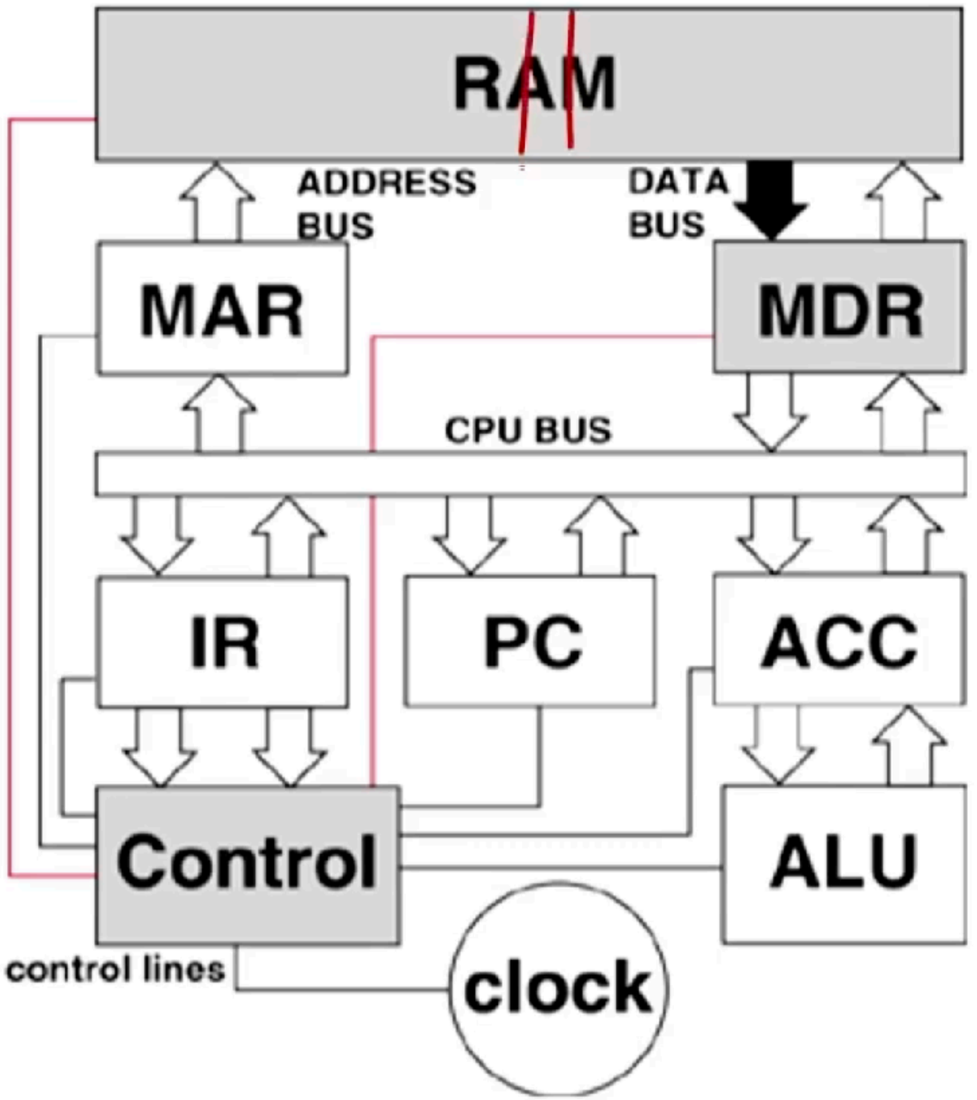
Instruction Fetch

- PC contains the address of next instr.
- Address transferred to MAR from PC (via internal bus)
- Instruction in located in the memory
-



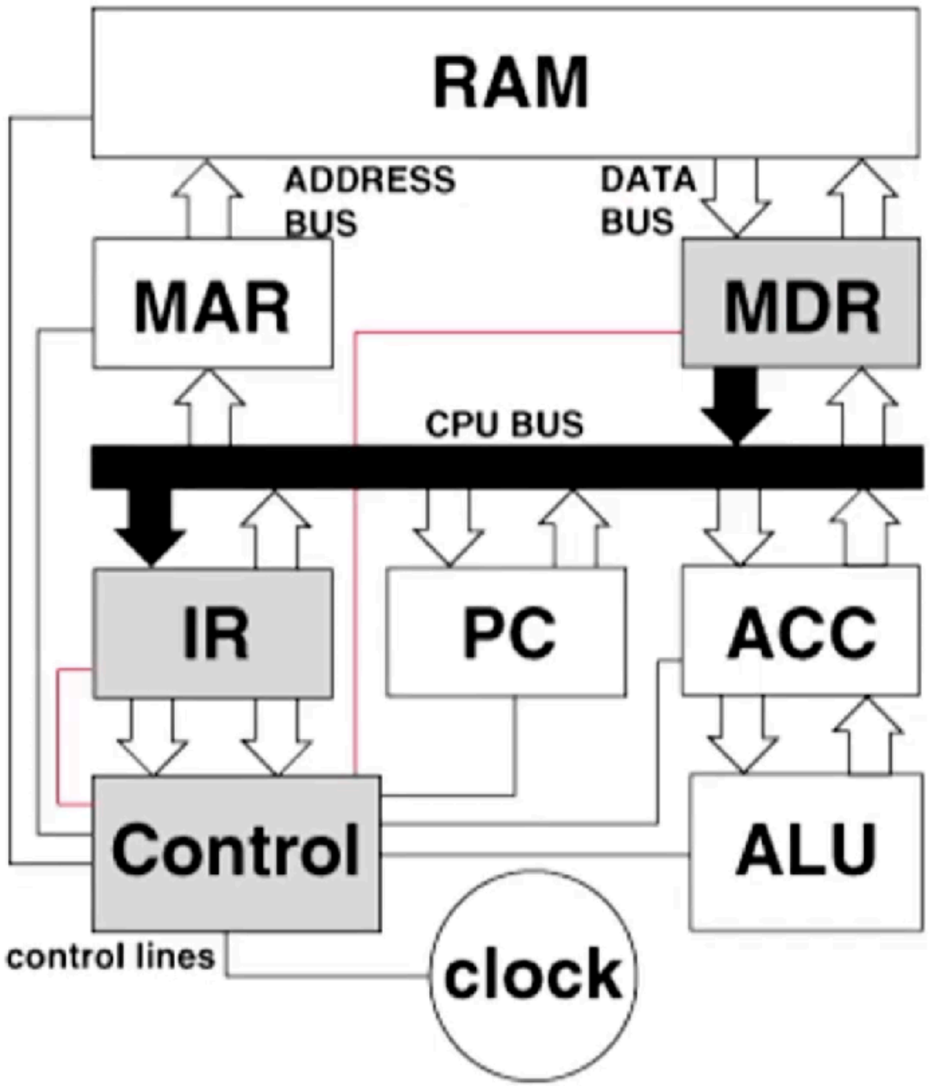
Instruction Fetch

- PC contains the address of next instr.
- Address transferred to MAR from PC (via internal bus)
- Instruction in located in the memory
- **Instruction in the memory address is copied to MDR**
-



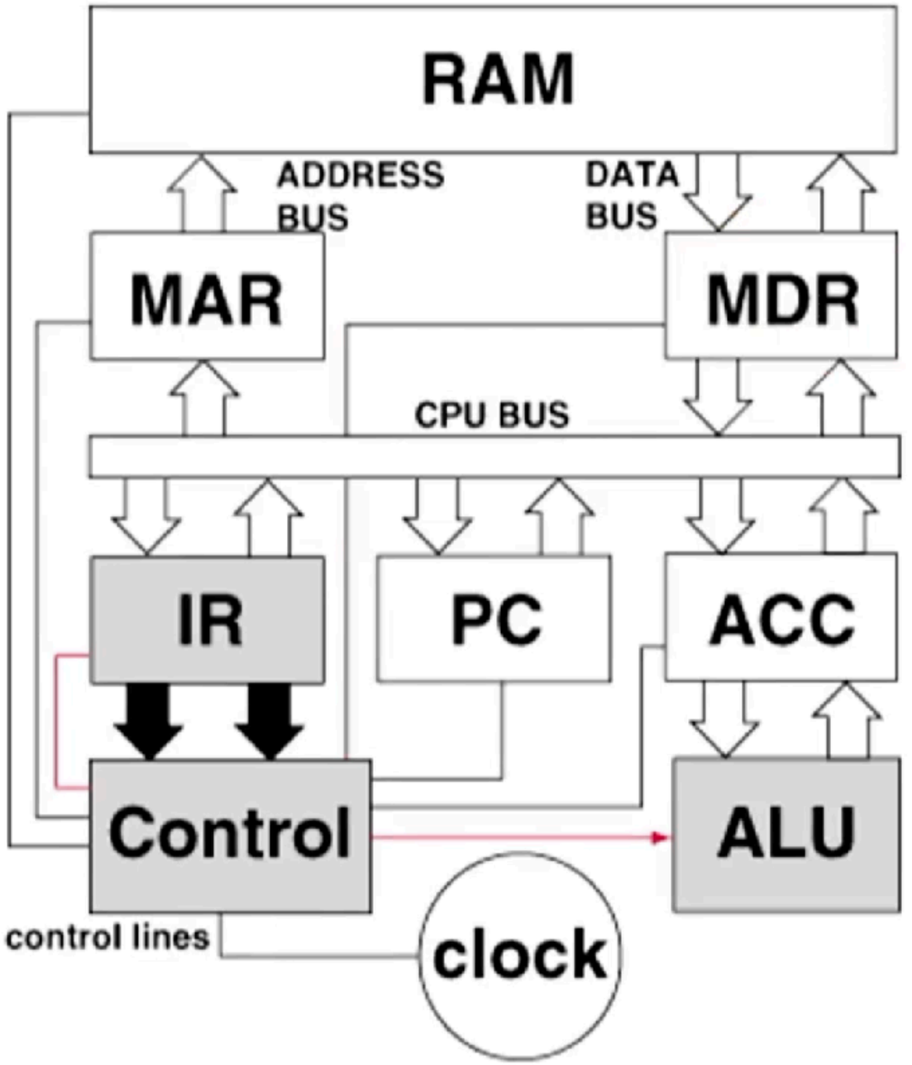
Instruction Decode

- PC contains the address of next instr.
- Address transferred to MAR from PC (via internal bus)
- Instruction in located in the memory
- Instruction in the memory address is copied to MDR
- **Instruction transferred from MDR to IR for decoding**



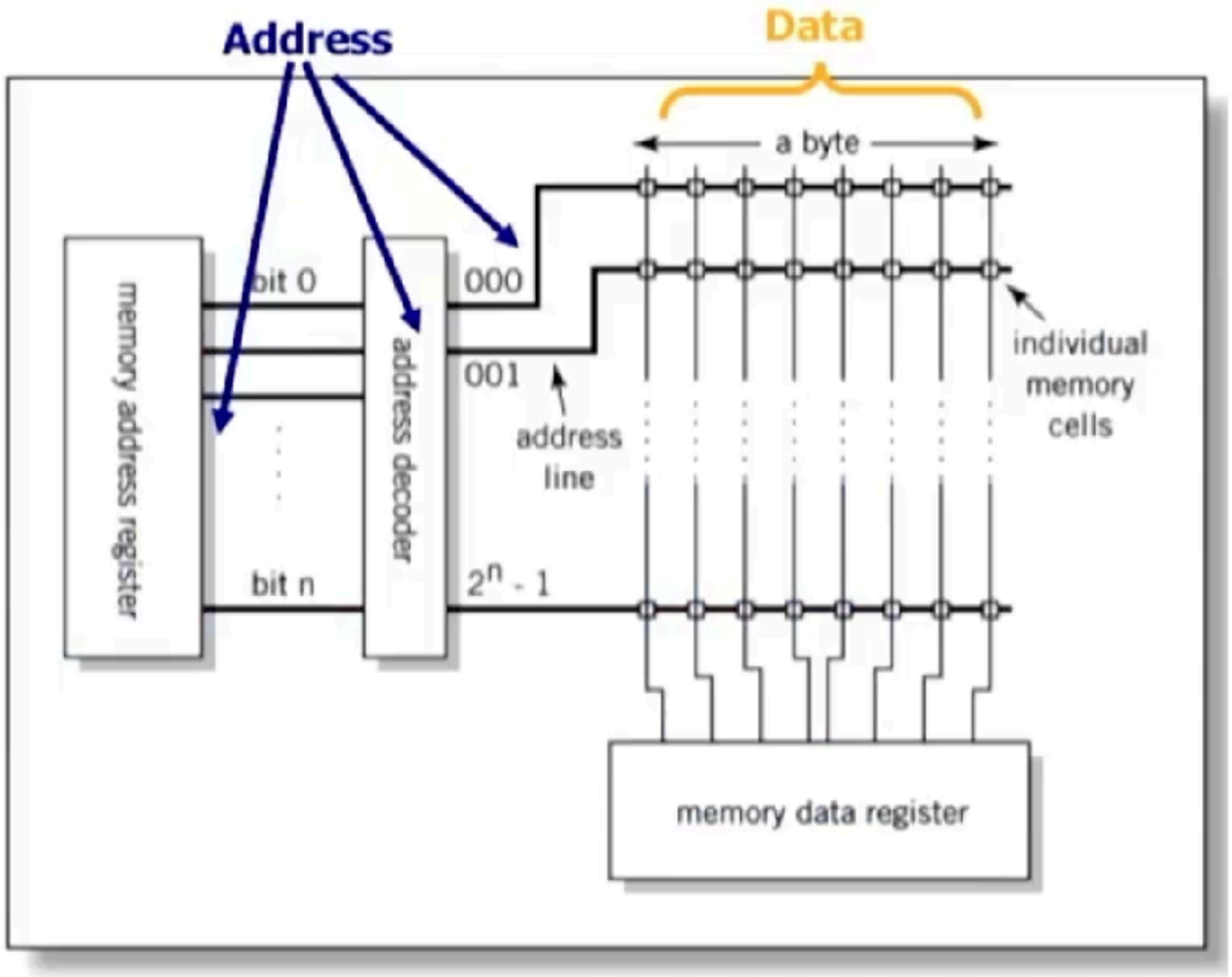
Instruction Execution

- PC contains the address of next instr.
- Address transferred to MAR from PC (via internal bus)
- Instruction in located in the memory
- Instruction in the memory address is copied to MDR
- Instruction transferred from MDR to IR for decoding
- **Control Unit signals appropriate devices to execute instruction**



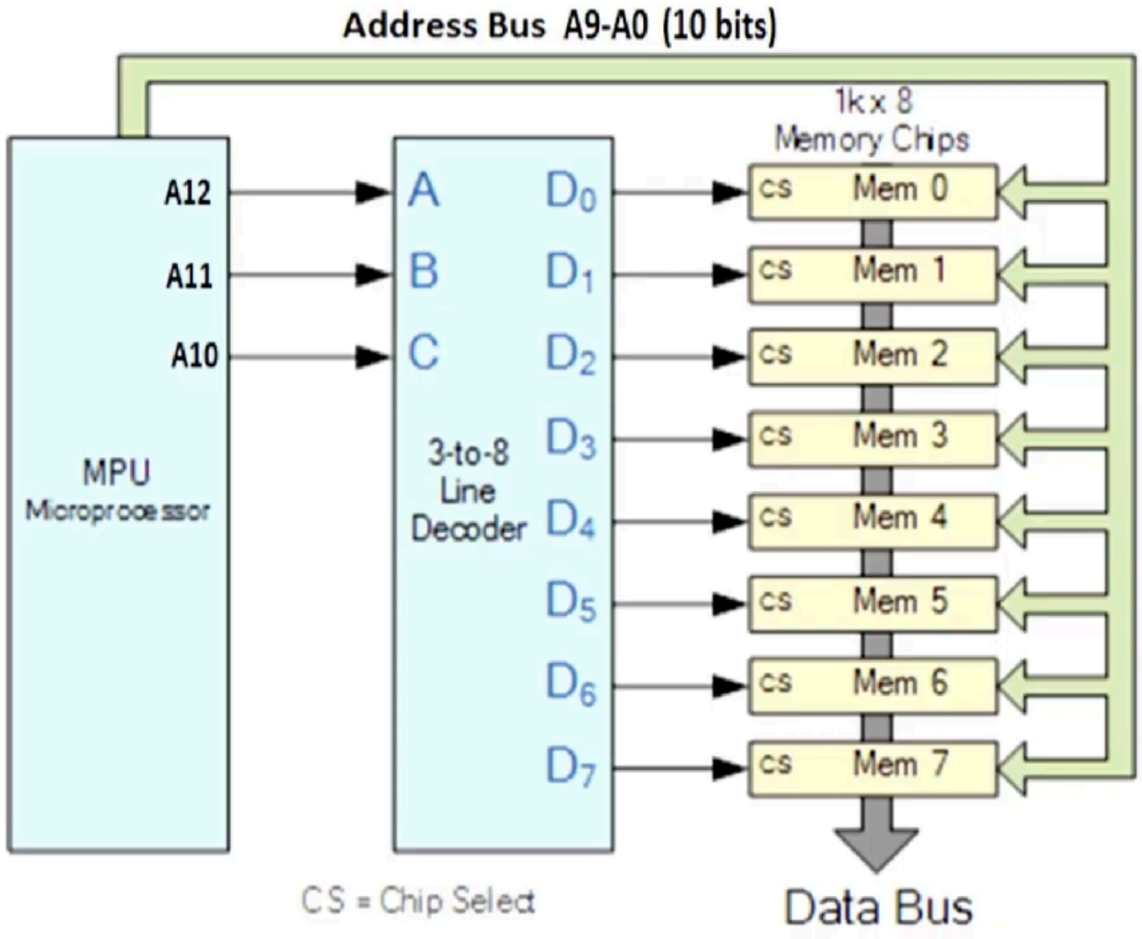
Instruction Fetch

- MAR is attached to an address decoder
- Decoder takes in n bits and chooses one of 2^n address lines
- One address line becomes “high”
- And the output of the decoder is fed into the MDR
- Once MAR has the address location and decoder is enabled to read the MDR has contents of the address



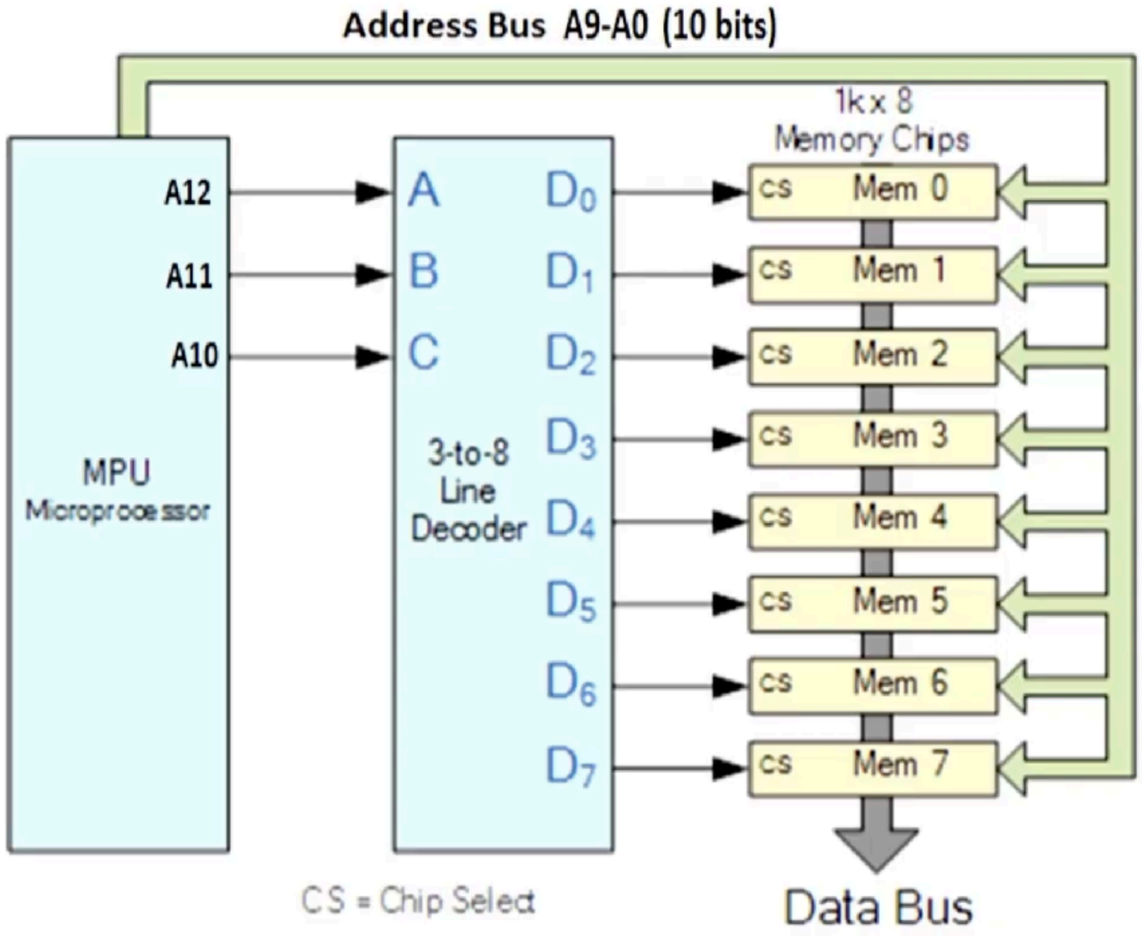
How to deal with large memory requirement?

- Suppose you have only 1 kB ICs
- And a 3 by 8 decoder
- Need 8 kB memory
- Needs 13 address bits
- Use the 3 MSBs as chip select and the other bits to specify the address inside the memory chip



How to deal with large memory requirement?

- Suppose you have only 1 kB ICs
- And a 3 by 8 decoder
- Need 8 kB memory
- Needs 13 address bits
- Use the 3 MSBs as chip select and the other bits to specify the address inside the memory chip
- **This is describing only first principles**



Languages of a computer

3 levels

- ISA is the vocabulary in the language of the computer
- The language of a computer can be written in three levels
- Machine language, which is computer readable
- Assembly language, which is human readable representation of machine language
- High level language, which is human readable
- Translators between different levels of language

Sequential Execution of Stored Program

Instruction Execution

- Instructions are stored in the memory
- In general, instruction lengths are word lengths
- Processor fetches, decodes, executes an instruction and proceeds to the next instruction
- Continues with the next instruction
- The address of next instruction is store in PC
- PC is incremented, typically by word length, PC ← PC + 4
- PC incrementer (circuit) is connected to the PC, and implements the increment

Types of Instructions

Broad classification

- Arithmetic and Logical
- Data movement
- Control Flow
- Can vary from processor to processor

Types of Instructions

Arithmetic and Logical Instructions

- Performed inside the Arithmetic and Logic Unit (ALU)
- Integer Arithmetic
- Comparison of two quantities
- Logical Operations - AND, OR, NOR, etc
- Shifting operations - right shift, left shift, rotating bits in a register
- Testing, comparison operations

Types of Instructions

Data Movement Instructions

- Moving data between memory hierarchy
- From memory to CPU registers [Load]
- From CPU registers to memory [Store]
- From memory to memory
- Input and Output [I/O]
-

Types of Instructions

Control Flow Instructions

- Start program
- Halt program
- Skip/jump to instructions
- Testing conditions whether to jump/skip instructions

Example Instructions

- add is the operation code
- b,c are the source operands
- a is the destination operand
-

High-level code

```
a = b + c;
```

Assembly

```
add a, b, c
```

Variable-Register Mapping

```
b = R1, c = R2, a = R0
```

Assembly Vs Machine Language

```
ADD  R0, R1, R2
1010 0000 0001 0010
```

Instruction Sets

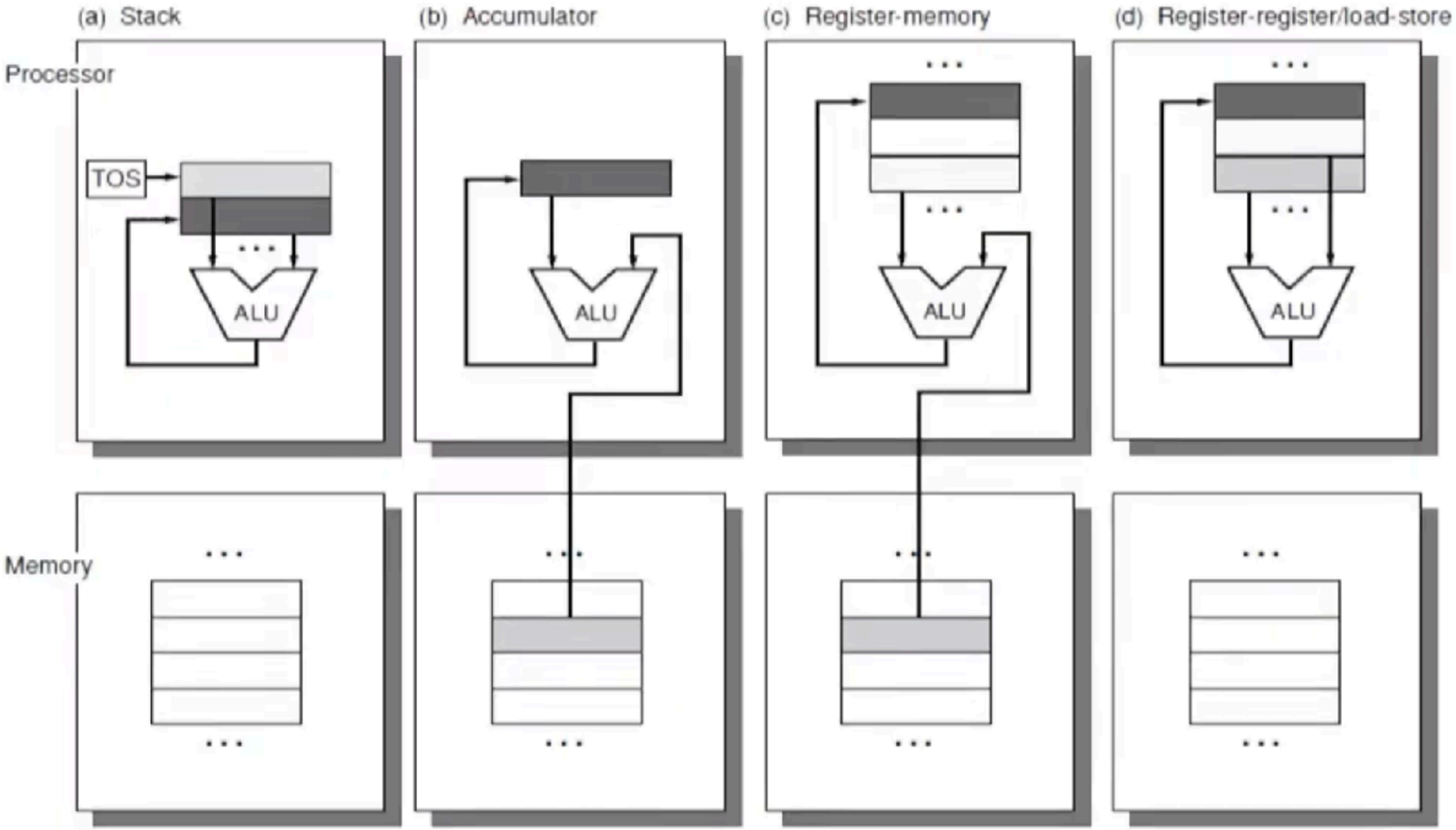
1. **Computer Instruction**: binary code that specifies a sequence of microoperations for the computer.
2. **Instruction Code**: a group of bits that instruct the computer to perform a specific operations.

Instruction Set Architecture vs Micro-architecture

- Application Program → ISA → Micro-arch. → Circuits
- ISA:
  - Hardware/Software Interface
  - The info needed by the programmer to write, debug programs
- Micro-Architecture:
  - Implementation of an ISA
  - Invisible to the software
- Micro-processor is ISA and Micro-architecture and the underlying circuits

Instruction Set Architecture - Classification

- Stack architecture
- Accumulator architecture
- Register-Memory architecture
- Register-Register or Load/Store architecture
- Instruction: Opcode Operands



Stack	Accumulator	Register (register-memory)	Register (load-store)
Push A	Load A	Load R1,A	Load R1,A
Push B	Add B	Add R3,R1,B	Load R2,B
Add	Store C	Store R3,C	Add R3,R1,R2
Pop C			Store R3,C

The code sequence for C = A + B for four classes of instruction sets.

Img. Source: Prof. John Jose

Instruction Set Architecture - Classification

- Stack architecture
- Accumulator architecture
- Register-Memory architecture
- Register-Register or Load/Store architecture
- Instruction: Opcode + Operands

ISA - Example: A = (B+C)*D + E

- *Stack Arch.*:
- Push D
- Push B
- Push C
- ADD
- MUL
- Push E
- Add
- Pop

- *ACC Arch*:
- Load B
- Add C
- Mul D
- Add E
- Store A

- *Load Store Arch:*
- Load R1, D
- Load R2, B
- Load R3, C
- Add R4, R2, R3
- Mul R5, R1, R4
- Load R6, E
- Add R7, R5, R6
- Store R7, A

Instruction Set

1. **Computer Instruction**: binary code that specifies a sequence of microoperations for the computer.
2. **Instruction Code**: a group of bits that instruct the computer to perform a specific operations.

- Two parts:

(a) **operation code** (specify the operations such as add, substract;
(b) **addresses**: registers/memory where results to be stored or registers/memory where operands are found

- **Each computer has its own instruction code format**

Instruction Set Architecture

3. Instruction Set Architecture (ISA)

– part of abstract model of a computer that defines how the CPU is controlled by the software.

– Serves as an interface between software and hardware, specifying both what the processor is capable of doing as well as how it gets done.

– Can be viewed as a programmer's manual because it's the portion visible to the assembly language programmer, the complier writer, and the application programmer.

– Also consists of the instruction set.

– Classic examples: IBM 360 series, Intel x86 series, etc.

Instruction Format

• An instruction consists of two parts:-

a) Operation Code or Opcode

• Specifies the operation to be performed by the instruction.

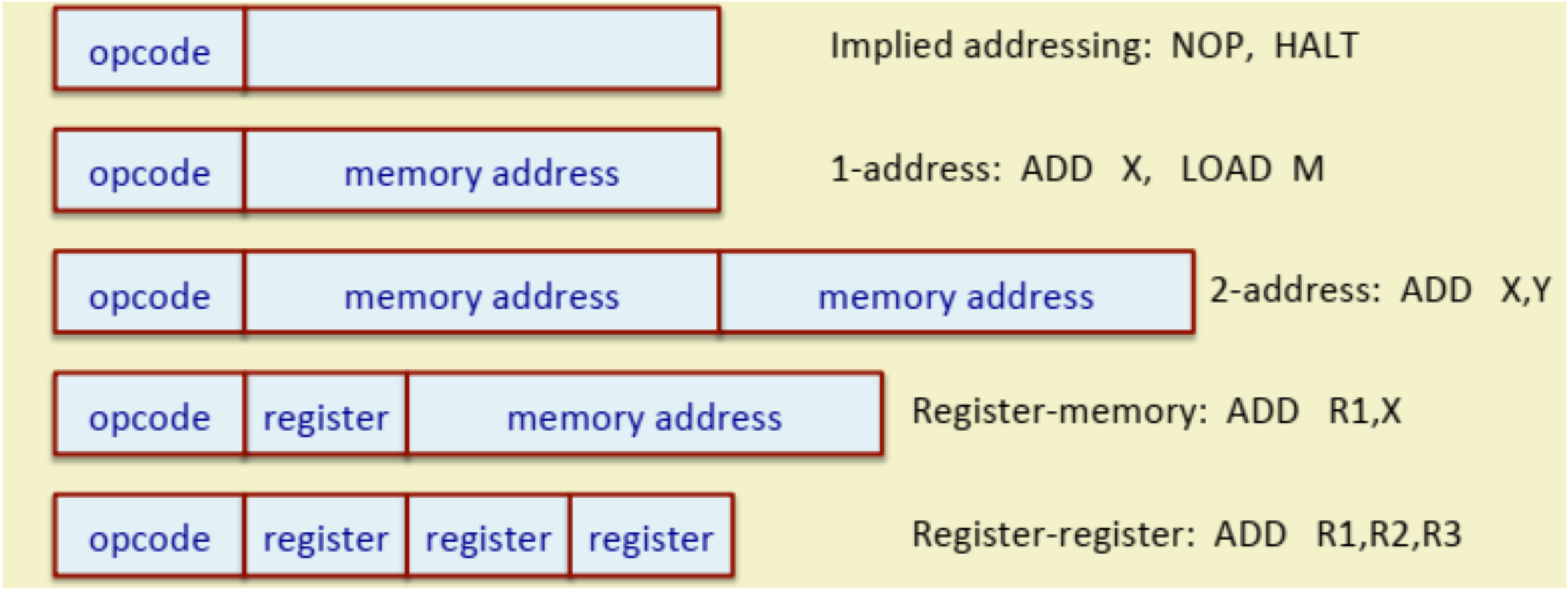
• Various categories of instructions: data transfer, arithmetic and logical, control, I/O and special
machine control.

b) Operand(s)

• Specifies the source(s) and destination of the operation.

• Source operand can be specified by an immediate data, by naming a register, or specifying the address
of memory.

Instruction Example



Instruction Format

- Destination can be specified by a register or memory address.
- Number of operands varies from instruction to instruction.

- Also for specifying an operand, various ***addressing modes*** are possible:
  - Implicit addressing
  - Immediate addressing
  - Direct addressing
  - Indirect addressing
  - Relative addressing
  - Indexed addressing, and many more.

Addressing Modes

• They specify the mechanism by which the operand data can be located.

•Some ISA's are quite complex and supports many addressing modes.

• ISA's based on load-store architecture are usually simple and support very limited number of addressing modes.

• Various addressing modes exist:
– Implicit, Immediate, Direct, Indirect, Register, Register Indirect, Indexed, Stack, Relative, Autoincrement, Autodecrement, Based, etc.
– Not all processors support all addressing modes.
– We shall briefly look at the common addressing modes and how they work.

Addressing Modes

• Various addressing modes exist:
– Implicit
– Immediate,
– Direct,
– Indirect,
– Register,
– Register Indirect,
– Indexed,
– Stack,
– Relative,
– Autoincrement/Autodecrement.

– Not all processors support all addressing modes.
– We shall briefly look at the common addressing modes and how they work.

Implicit Addressing

- The operand is not specified, but, implied.
  - No memory reference is required to access the operand.
  - Fast but limited number of operations possible.

- Examples:
  - CMA // Complement Accumulator
  - SLCA // Shift left and Count Accumulator

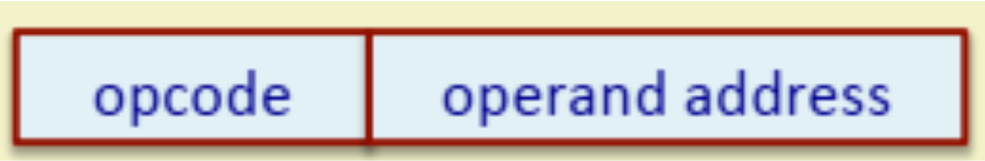
Immediate Addressing

- The operand is part of the instruction itself.
  - No memory reference is required to access the operand.
  - Fast but limited range (because a limited number of bits are provided to specify the immediate data).

- Examples:
  - ADD #25 // ACC = ACC + 25
  - MOVI R1 45 // Move the data 45H immediately to register R1

Direct Addressing

- .The instruction contains a field that holds the memory address of the operand.

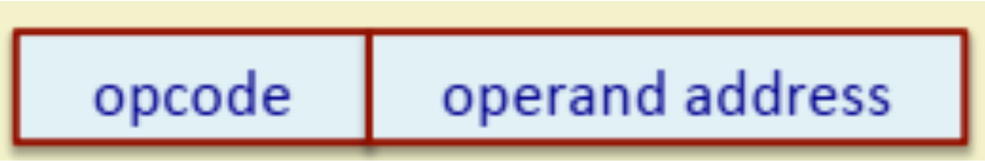


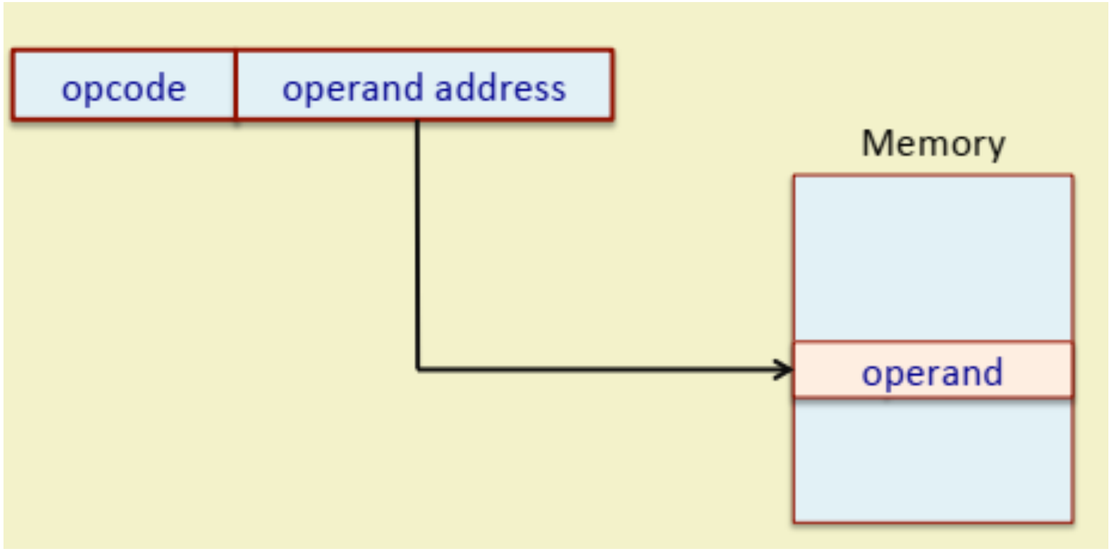
- Examples:
  - ADD 20A6H // ACC = ACC + Mem[20A6]

- Single memory access is required to access the operand.
  - No additional calculations required to determine the operand address.
  - Limited address space (as number of bits is limited, say, 16 bits).

Direct Addressing

- .The instruction contains a field that holds the memory address of the operand.





- Examples:
  - ADD 20A6H // ACC = ACC + Mem[20A6]

- Single memory access is required to access the operand.
  - No additional calculations required to determine the operand address.
  - Limited address space (as number of bits is limited, say, 16 bits).

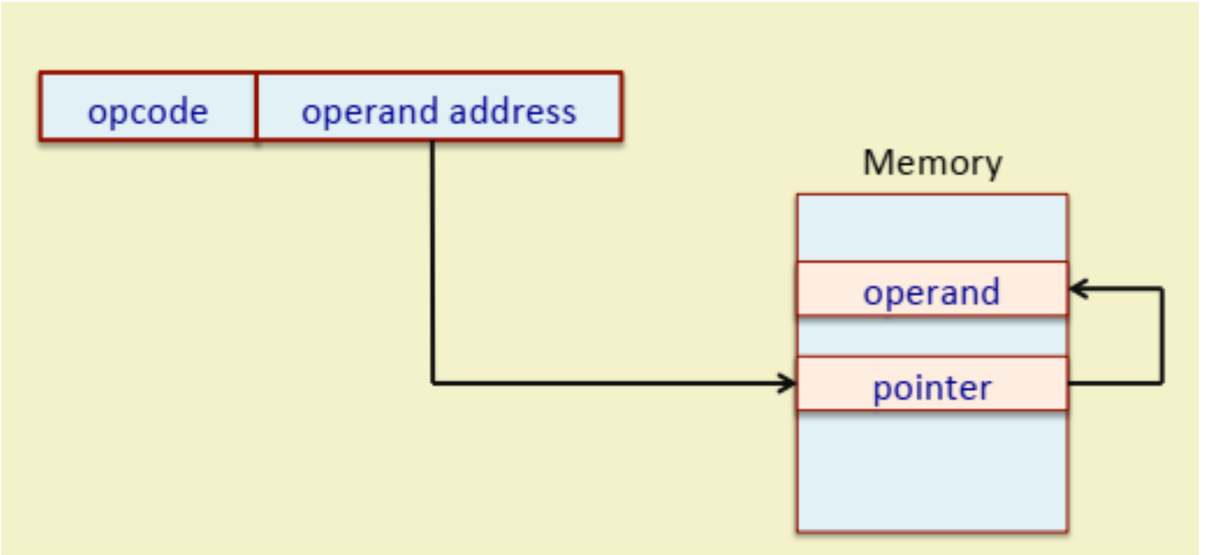
Indirect Addressing

• The instruction contains a field that holds the memory address, which in turn holds the memory address of the operand.

• Two memory accesses are required to get the operand value.

• Slower but can access large address space.
  – Not limited by the number of bits in operand address like direct addressing.

• Examples:
– ADD (20A6H) // ACC = ACC + (Mem[20A6])

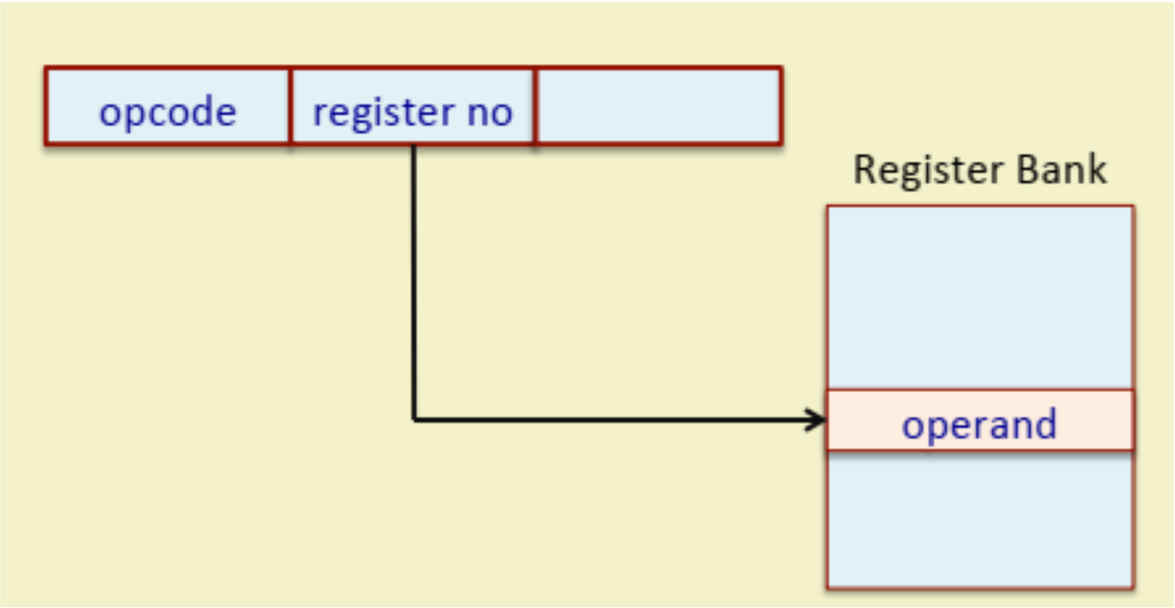


Register Addressing

- The operand is held in a register, and the instruction specifies the register number.
  - Very few number of bits needed, as the number of registers is limited.
  - Faster execution, since no memory access is required for getting the operand.

- Modern load-store architectures support large number of registers.

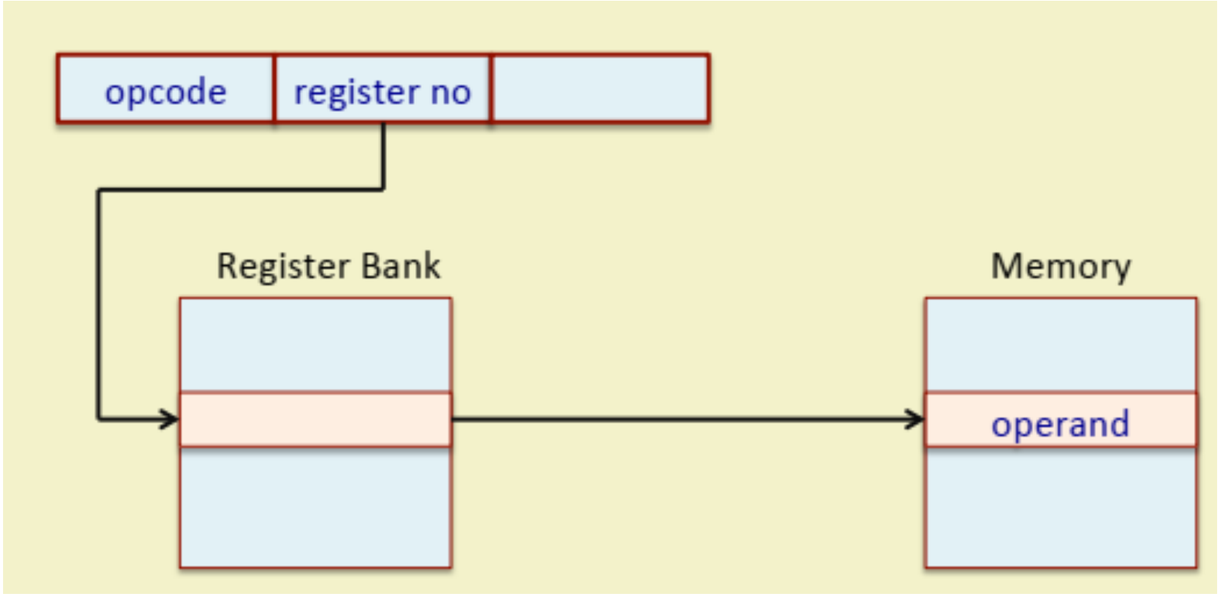
- Examples:
  - ADD R1, R2, R3 // R1 = R2 + R3
  - MOV R2, R5 // R2 = R5



Register Indirect Addressing

• The instruction specifies a register, and the register holds the memory address where the operand is stored.

• Example:
– ADD R1, (R5) // R1 = R1 + Mem[R5]

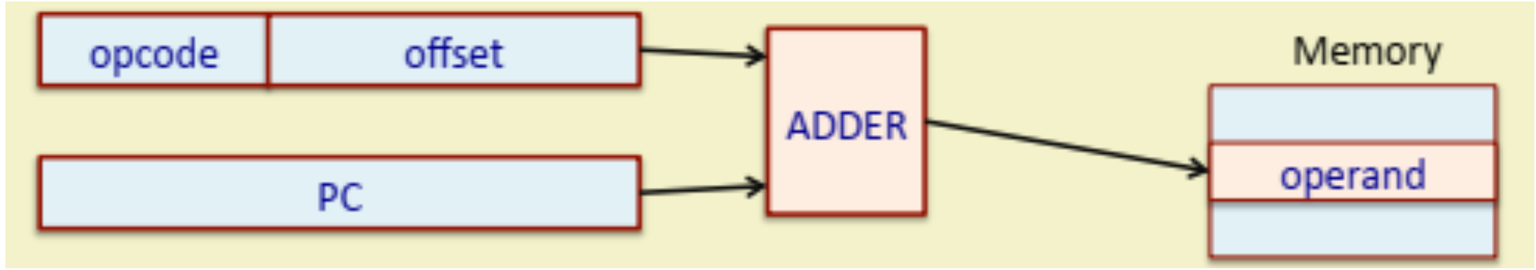


Relative Addressing (PC Related)

- The instruction specifies an offset of displacement,

  which is added to the program counter (PC) to get the effective address of the operand.

  – Since the number of bits to specify the offset is limited, the range of relative addressing is also limited.

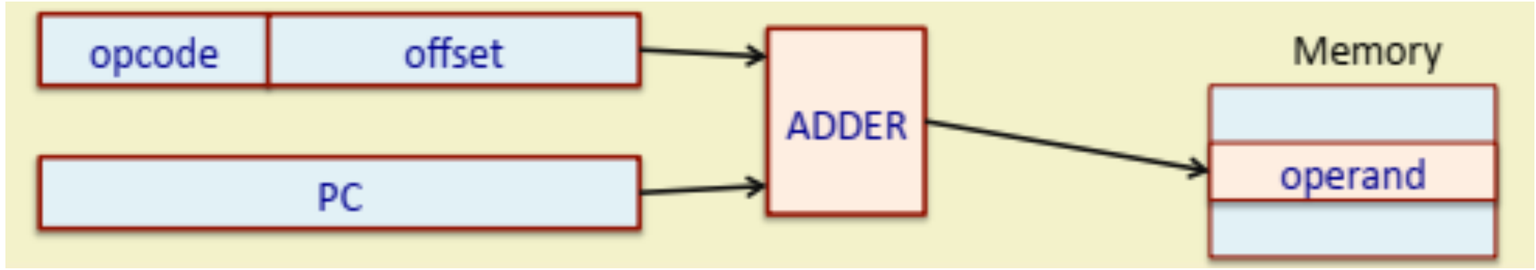


Relative Addressing (PC Related)

• The instruction specifies an offset of displacement, which is added to the program counter (PC) to get the effective address of the operand.

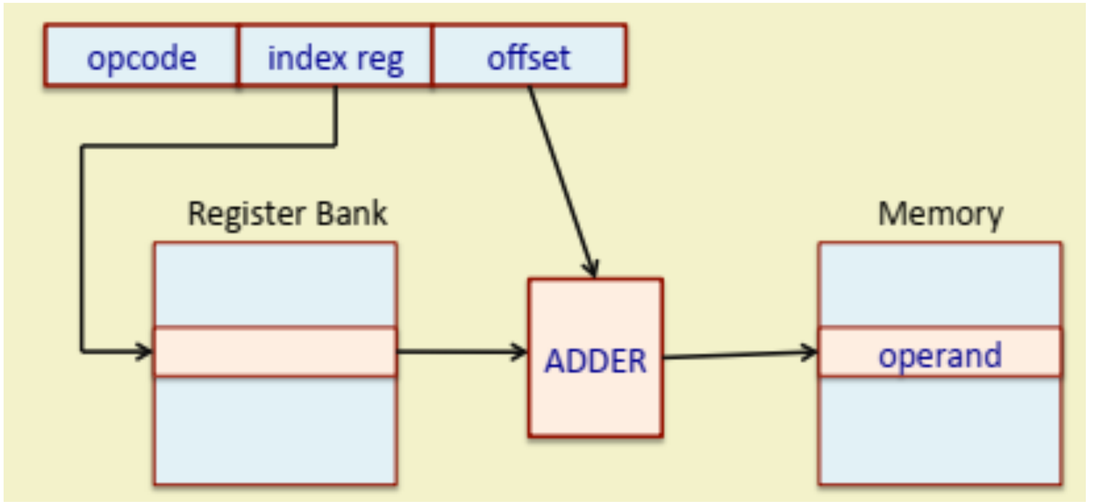
– Since the number of bits to specify the offset is limited, the range of relative addressing is also limited.

— Used for loop control, branching/jumping instructions



Indexed Addressing

• Either a special-purpose register, or a general-purpose register, is used as *index register* in this addressing
mode.
• The instruction specifies an offset of displacement, which is added to the index register to get the effective address of the operand.
• Example:
– LOAD R1, 1050(R3) // R1 = Mem[1050+R3]
• Can be used to sequentially access the elements of an array.
– Offset gives the starting address of the array, and the index register value specifies the array elementto be used



Stack Addressing

- Operand is implicitly on top of the stack.

- Used in zero-address machines earlier.

- Examples:
  - ADD
  - PUSH X
  - POP X

- Many processors have a special register called the stack pointer (SP) that keeps track of the stack-top in memory.

  - PUSH, POP, CALL, RET instructions automatically modify SP.

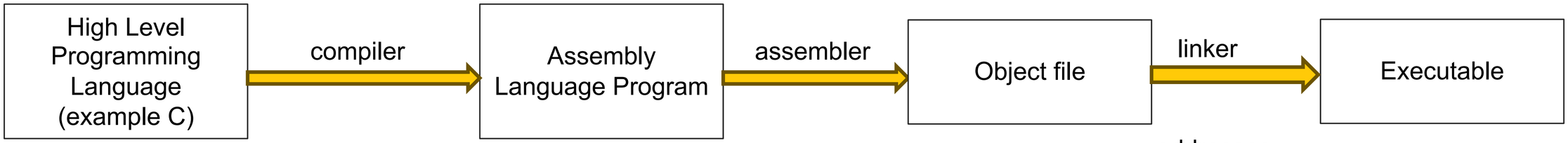
Other Addressing Modes

• Base addressing
– The processor has a special register called the *base register* or *segment register*.
– All operand addresses generated are added to the base register to get the final memory address.
– Allows easy movement of code and data in memory.

• Autoincrement and Autodecrement
– The register holding the operand address is automatically incremented or decremented after accessing the operand (like a++ and a-- in C).

• Implied/Implicit
– the operand is hidden and the data to be operated is available in the instruction itself.
- RLC (rotate accumulator A left by one bit) R1, (R5)

Transformation of Code




```
 3 void increment_elements(int *array, int n){
 4     int i;
 5     for(i=0; i<n; ++i){
 6         array[i] = array[i] + 1;
 7     }
 8 }
 9
10 void exor_elements(int *array, int n){
11     int i;
12     for(i=0; i<n; ++i){
13         array[i] = array[i] ^ 0xFFFFFFFF ;
14     }
15 }
16
17 int main(){
18     int A[10];
19     int i;
20     int choice;
21
22     for(i=0; i<10; ++i) scanf("%d", &A[i]);
23     if (A[0] == 0 && A[1]==0){
24         increment_elements(&A[2], 8);
25     }
26     else{
27         exor_elements(&A[2], 8);
28     }
29 }
```

C Code

```
addiw    s0,s0,-16
unimp
ld       s0,32(a2)
lui      t1,0xffffa
bgeu     zero,a6,14 <
c.slli   zero,0xa
unimp
addi     s1,sp,32
lui      tp,0xfffe1
fld      fa2,376(sp)
ld       a3,80(a0)
0x7032
lw       a2,120(a4)
addiw    tp,tp,-5
fld      fa2,224(s0)
0x5f307032615f
fld      ft4,120(sp)
fld      fa2,224(s0)
0x5f307032645f
0x30703263
```

Assembly code

```
0000 797122f4 0018233c a4fcae87 232af4fc
0010 232604fe 35a08327 c4fe8a07 033784fd
0020 ba979843 8327c4fe 8a078336 84fdb697
0030 05270127 98c38327 c4fe8527 2326f4fe
0040 0327c4fe 832744fd 01278127 e345f7fc
0050 01000100 22744561 82807971 22f40018
0060 233ca4fc ae87232a f4fc2326 04fe05a8
0070 8327c4fe 8a070337 84fdba97 94438327
0080 c4fe8a07 033784fd ba973687 1347f7ff
0090 012798c3 8327c4fe 85272326 f4fe0327
00a0 c4fe8327 44fd0127 8127e343 f7fc0100
00b0 01002274 45618280 397106fc 22f88000
00c0 232604fe 2da01307 04fc8327 c4fe8a07
00d0 ba97be85 b7070000 13850700 97000000
00e0 e7800000 8327c4fe 85272326 f4fe8327
00f0 c4fe1b87 0700a547 e3d7e7fc 832704fc
0100 91ef8327 44fc99eb 930704fc a107a145
0110 3e859700 0000e780 000011a8 930704fc
0120 a107a145 3e859700 0000e780 00008147
```

Object file
(relocatable. Addresses start from 0)

addresses

```
100b0 93070000 99c71735 00001305 e5aa6f20
100c0 30008280 97410100 9381c1ed 1385a176
100d0 17460100 1306066d 098e8145 ef000023
100e0 17250000 1305057e 19c51735 00001305
100f0 a5a7ef20 e07cef00 c01a0245 2c000146
10100 ef00e010 41a24111 22e01384 017a8347
10110 040006e4 99ef9307 000089cb 17350100
10120 1305c53e 97000000 e7000000 85472300
10130 f400a260 02644101 82809307 000099cb
10140 9385817a 17350100 1305453c 17030000
10150 67000000 82807971 22f40018 233ca4fc
10160 ae87232a f4fc2326 04fe35a0 8327c4fe
10170 8a070337 84fdba97 98438327 c4fe8a07
10180 833684fd b6970527 012798c3 8327c4fe
10190 85272326 f4fe0327 c4fe8327 44fd0127
101a0 8127e345 f7fc0100 01002274 45618280
101b0 797122f4 0018233c a4fcae87 232af4fc
101c0 232604fe 05a88327 c4fe8a07 033784fd
101d0 ba979443 8327c4fe 8a070337 84fdba97
```

Executable
(addresses resolved)

Encoded instructions

Instruction Cycle Tradeoff

- Objective 1: Minimize the number of cycles per instruction.
- Objective 2: Minimize the the number of instruction per program.
- **Trade off between these two objectives**

RISC vs CISC

- RISC - Reduced Instruction Set Computer
- Aim to reduce the cycles per instruction
- Simple Instructions and Encoding
- Fewer Addressing modes
- More instruction for a task
- Larger code base

- CISC - Complex Instruction Set Computer
- Aim to reduce the number of instruction per program
- Instructions of varying complexity
- Lesser number of instruction per task
- Smaller code base

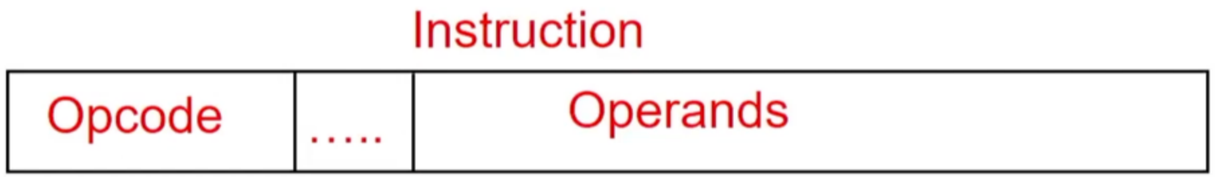
CISC vs RISC

- Emphasis on Hardware
- Multiple Instruction sizes and formats
- More addressing modes
- Use of microprogramming to break instruction into subtask
- Instructions take varying amounts of time
- Pipeling is difficult

- Emphasis on Software
- Fixed instruction size and fewer formats
- Less addressing modes
- Uses complex compiler to break down higher level task to simple instructions
- One cycle per instruction (average)
- Pipelining is easy

Instuction Encoding

Instruction Format



- Layout of bits in an instruction
- Includes opcode
- Includes (implicit or explicit) operand(s)
- Usually more than one instruction format in an instruction set

Instruction Encoding

Bit Allocation in Instruction Format

- # of Addressing Modes
- # of Operands
- Operand from register or memory?
- # of registers
- # Ranges of Addresses
- “Word” size (address granularity)

Instruction Encoding

Bit Allocation in Instruction Format

- # of Addressing Modes
- # of Operands
- Operand from register or memory?
- # of registers
- # Ranges of Addresses
- “Word” size (address granularity)
- 10010010101010011100010010001010

Instruction Encoding

Bit Allocation in Instruction Format

- # of Addressing Modes
- # of Operands
- Operand from register or memory?
- # of registers
- # Ranges of Addresses
- “Word” size (address granularity)
- 10010010101010011100010010001010
- 11110000001010101100010010001010

Instruction Length

- Depends on
  - Memory size
  - Memory organization
  - Bus size
  - Complexity of CPU

Numerical Example 1

A processor has 16 distinct instructions (unique opcodes) and 32 general purpose registers. A 24-bit instruction word has an opcode, two registers operands and an immediate operand. How many bits are possible for the immediate operand field?

Numerical Example 1

A processor has 16 distinct instructions (unique opcodes) and 32 general purpose registers. A 24-bit instruction word has an opcode two registers operands and an immediate operand. How many bits are possible for the immediate operand field?

16 opcodes: 4 bits for opcode field

32 GPR : 5 bits to address them

Instruction (24 bits)

Opcode	Operand1	Operand2	Imm. Operand
4	5	5	?

Numerical Example 2

- A RISC processor has 16 general purpose registers. It operates on 16-bit instruction and has broadly three type of instructions: register type, memory type and branch type. The register type instruction has 3 register operands and memory type instruction has an 8-bit memory offset and a single register operand. The branch instructior has 2 register operands. If there are 8 unique opcodes for register type instructions, and 6 unique opcodes for memory type instruction, what is the maximum number of unique opcodes possible for branch. type instructions? Illustrate an instruction encoding pattern.

Numerical Example 2

- A RISC processor has 16 general purpose registers. It operates on 16-bit instruction and has broadly three type of instructions: register type, memory type and branch type. The register type instruction has 3 register operands and memory type instruction has an 8-bit memory offset and a single register operand. The branch instructior has 2 register operands. If there are 8 unique opcodes for register type instructions, and 6 unique opcodes for memory type instruction, what is the maximum number of unique opcodes possible for branch. type instructions? Illustrate an instruction encoding pattern.
- 16 GPRs - 4 bits
- 16-bit instruction: R-Type, M-Type & B-Type
- R-Type: 4-R+4-R+4-R (8 opcodes), M-Type: 4-R+8-M (6 opcodes)

Numerical Example 2

- A RISC processor has 16 general purpose registers. It operates on 16-bit instruction and has broadly three type of instructions: register type, memory type and branch type. The register type instruction has 3 register operands and memory type instruction has an 8-bit memory offset and a single register operand. The branch instructior has 2 register operands. If there are 8 unique opcodes for register type instructions, and 6 unique opcodes for memory type instruction, what is the maximum number of unique opcodes possible for branch. type instructions? Illustrate an instruction encoding pattern.
- 16 GPRs - 4 bits
- 16-bit instruction: R-Type, M-Type & B-Type
- R-Type: 4-R+4-R+4-R (8 opcodes), M-Type: 4-R+8-M (6 opcodes)\
- M-Type: ?

Numerical Example 2

- 16 GPRs - 4 bits
- 16-bit instruction: R-Type, M-Type & B-Type
- R-Type: 4-R+4-R+4-R (8 opcodes), M-Type: 4-R+8-M (6 opcodes)
- M-Type: ?
-

R-Type	0 X X X	R (4)	R (4)	R (4)	0000 to 0111 (8 opcodes)
M-Type	1 X X X	R (4)	M (8)		1000 to 1101 (6 opcodes)

-

B-Type	1 1 1 0 1 1 1 1	XXXX	R (4)	R (4)	1110XXXX (16 opcodes) + 1111XXXX (16 opcodes) = 32

-

Numerical Example 3

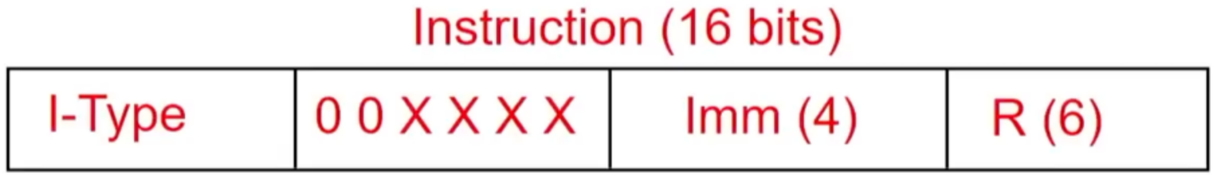
- A processor has 64 registers and it uses a 2-byte instruction format for all its instructions. It has two types of instruction. I-type and R-type. The I-type instruction contains an opcode, a register name and a 4-bit immediate value. The R-type instruction contains an opcode, and two register names. If there are 16 distinct opcodes for I-type instruction, then how many distinct opcodes are possible for R-type instruction? Illustrate the instruction encoding pattern.

Numerical Example 3

- A processor has 64 registers and it uses a 2-byte instruction format for all its instructions. It has two types of instructions. I-type and R-type. The I-type instruction contains an opcode, a register name and a 4-bit immediate value. The R-type instruction contains an opcode, and two register names. If there are 16 distinct opcodes for I-type instruction, then how many distinct opcodes are possible for R-type instruction? Illustrate the instruction encoding pattern.

- 64 GPR = 6 bits.
- 16 bit instruction: I-Type & R-Type
- I Type: opcode + 6-R + 4-Immediate (16 opcodes)
- R-Type: opcode+ 6-R + 6-R (max opcodes = ?)

Numerical Example 3

- 64 GPR = 6 bits.
- 16 bit instruction: I-Type & R-Type
- I Type: opcode + 6-R + 4-Immediate (16 opcodes)
- R-Type: opcode+ 6-R + 6-R (max opcodes = ?)
- 

-

Numerical Example 3

- 64 GPR = 6 bits.
- 16 bit instruction: I-Type & R-Type
- I Type: opcode + 6-R + 4-Immediate (16 opcodes)
- R-Type: opcode+ 6-R + 6-R (max opcodes = ?)
-

Instruction (16 bits)

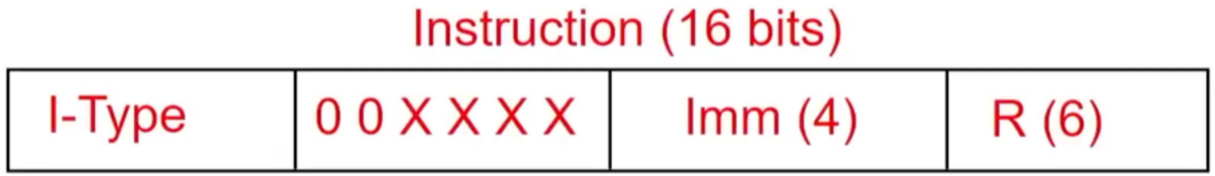
I-Type	0 0 X X X X	Imm (4)	R (6)

-

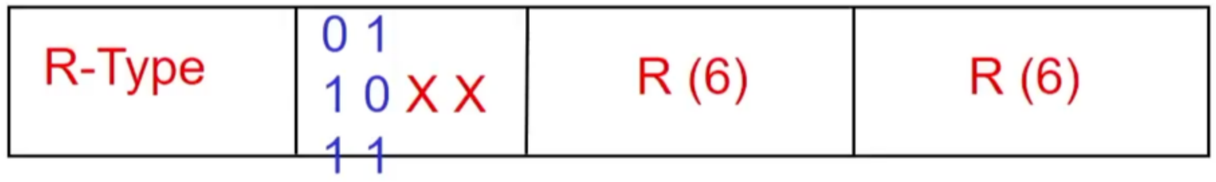
R-Type	0 1 1 0 X X 1 1	R (6)	R (6)

Numerical Example 3

- 64 GPR = 6 bits.
- 16 bit instruction: I-Type & R-Type
- I Type: opcode + 6-R + 4-Immediate (16 opcodes)
- R-Type: opcode+ 6-R + 6-R (max opcodes = ?)
-



-



XX= 00 to 11 (4 ops)
TOTAL = 3X4 =12 opcodes

Numerical Example 4

- Consider the case of a processor with 32 general purpose registers and an instruction length of 16-bits. The processor has a main memory (RAM) addressability of 256Bytes. It has 32 Type -1 instructions each with two register operands, two Type-ll instructions each with a register and a memory operand, and N type III instruction each with one register operand.
- What is the maximum possible value of N? Show an instruction encoding format and its interpretation.

Numerical Example 4

- Consider the case of a processor with 32 general purpose registers and an instruction length of 16-bits. The processor has a main memory (RAM) addressability of 256Bytes. It has 32 Type -1 instructions each with two register operands, two Type-ll instructions each with a register and a memory operand, and N type III instruction each with one register operand.
- What is the maximum possible value of N? Show an instruction encoding format and its interpretation.
-

16 bit Instruction

I	0	XXXXX (5 bits) (32 COMBINATIONS)	5 Bit Reg name	5 Bit Reg name

Numerical Example 4

- Consider the case of a processor with 32 general purpose registers and an instruction length of 16-bits. The processor has a main memory (RAM) addressability of 256Bytes. It has 32 Type -1 instructions each with two register operands, two Type-ll instructions each with a register and a memory operand, and N type III instruction each with one register operand.
- What is the maximum possible value of N? Show an instruction encoding format and its interpretation.
-

16 bit Instruction

I	0	XXXXX (5 bits) (32 COMBINATIONS)	5 Bit Reg name	5 Bit Reg name

-

II	1	00 /01 (2)	5 Bit Reg name	8 Bit Memory Address

Numerical Example 4

- Consider the case of a processor with 32 general purpose registers and an instruction length of 16-bits. The processor has a main memory (RAM) addressability of 256Bytes. It has 32 Type -1 instructions each with two register operands, two Type-ll instructions each with a register and a memory operand, and N type III instruction each with one register operand.
- What is the maximum possible value of N? Show an instruction encoding format and its interpretation.

16 bit Instruction

I	0	XXXXX (5 bits) (32 COMBINATIONS)	5 Bit Reg name	5 Bit Reg name

II	1	00 /01 (2)	5 Bit Reg name	8 Bit Memory Address

III	1	10 11	XXXXXXXX (8 bits) (512 COMBINATIONS)	5 Bit Reg name

Thank you !