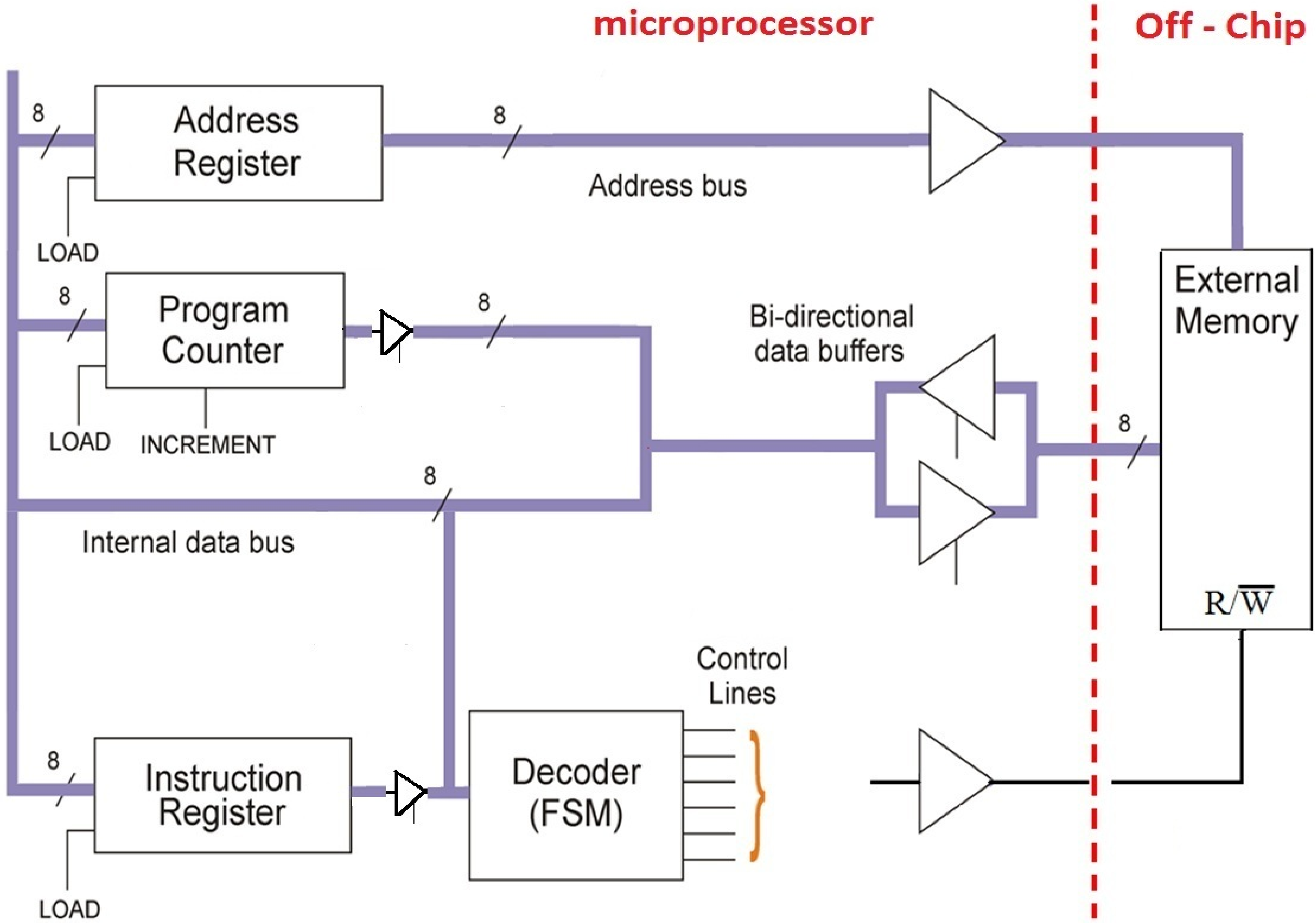
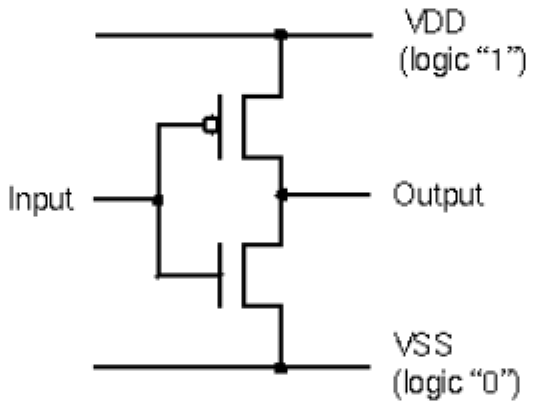
CONTROL UNIT

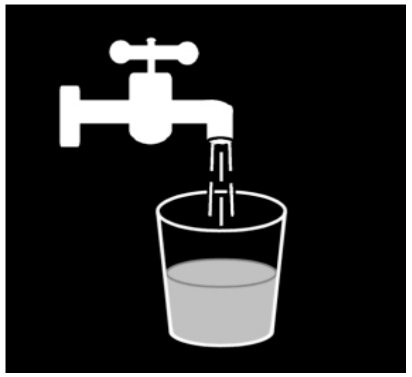
- Control Components
- Hard Wired Control
- Micro-Programmed Control
- Vertical and Horizontal Micro-coding
- Reduced Instruction Set Computer (RISC)

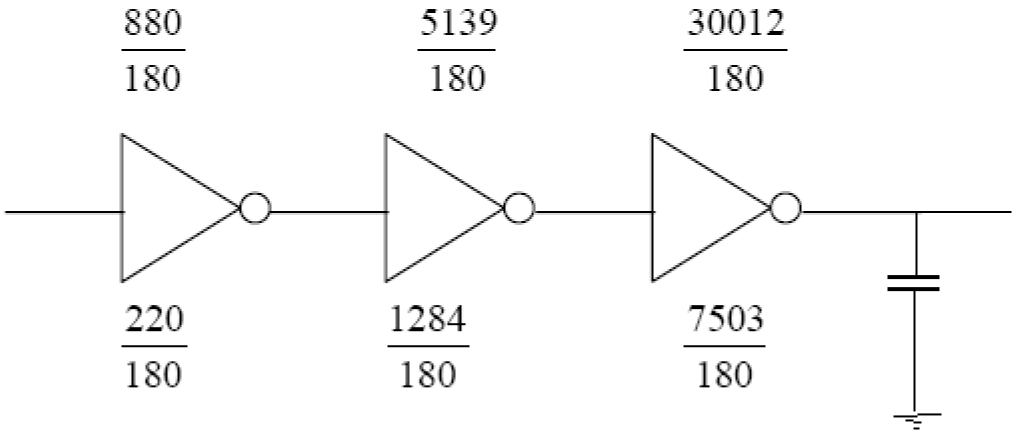


microprocessor
Off - Chip
8
Address Register
8
Address bus
LOAD
External Memory
8
Program Counter
8
Bi-directional data buffers
LOAD
INCREMENT
8
8
Internal data bus
R/$\overline{W}$
Control Lines
8
Instruction Register
Decoder (FSM)
LOAD

Output Buffer

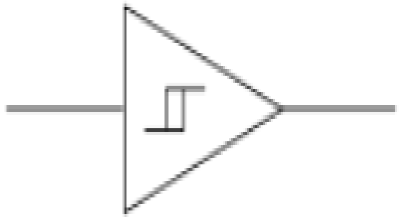




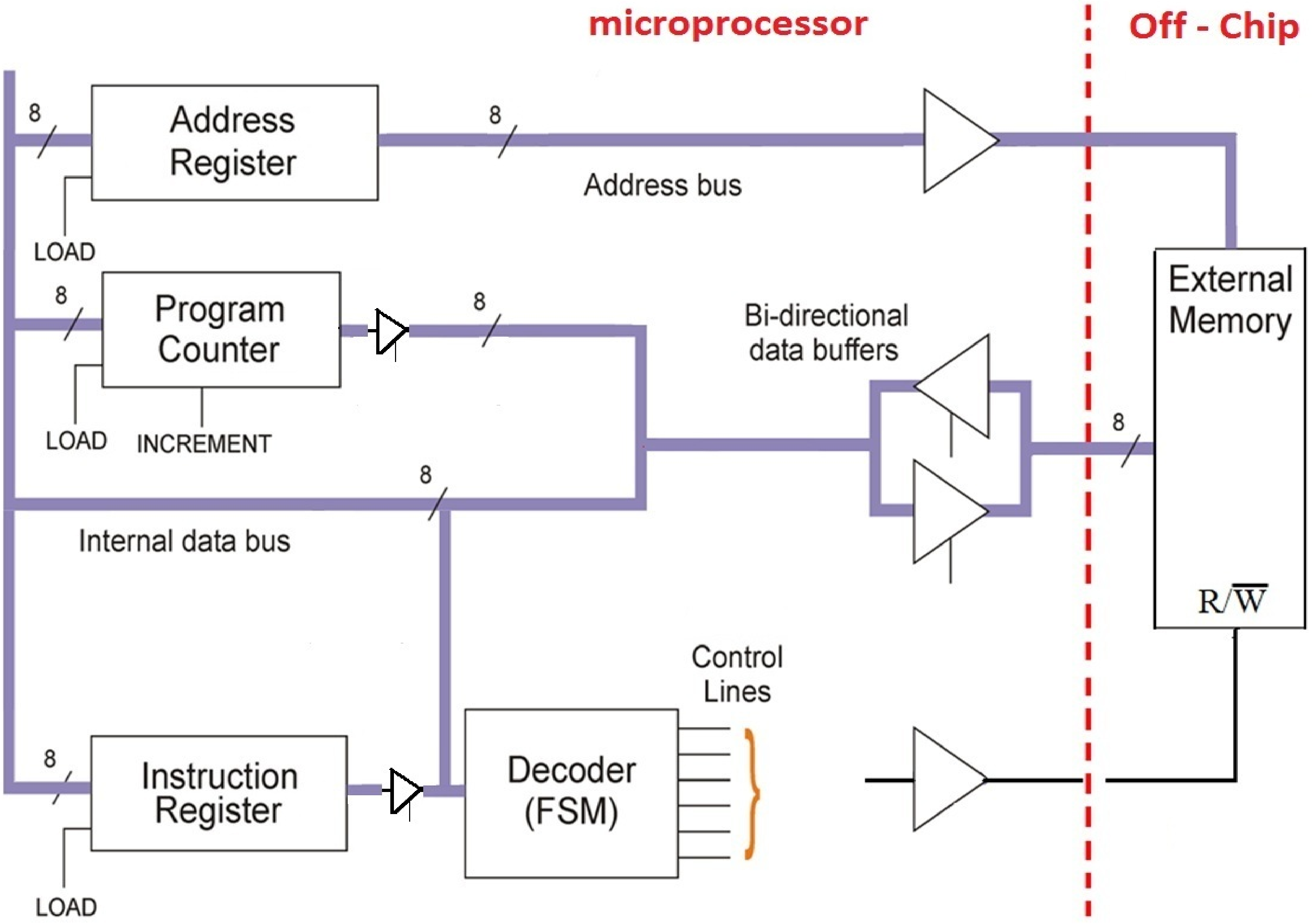


Input Buffer

Schmitt trigger - provides some noise immunity



the output retains its value until the input changes sufficiently to trigger a change


microprocessor
Off - Chip
Address Register
8
8
Address bus
LOAD
Program Counter
8
8
LOAD
INCREMENT
Bi-directional data buffers
External Memory
8
8
Internal data bus
R/$\overline{W}$
Control Lines
Instruction Register
8
Decoder (FSM)
LOAD

Instruction Register (IR)

Must hold instruction over the multiple cycles to draw on instruction information throughout instruction execution

An instruction requires two steps:

- *Instruction fetch* – obtaining an instruction from memory
- *Instruction execution* – the execution of a sequence of micro-operations to perform instruction processing
- Due to the use of the IR, these two steps require a minimum of two clock cycles

Memory Address Register (MAR)

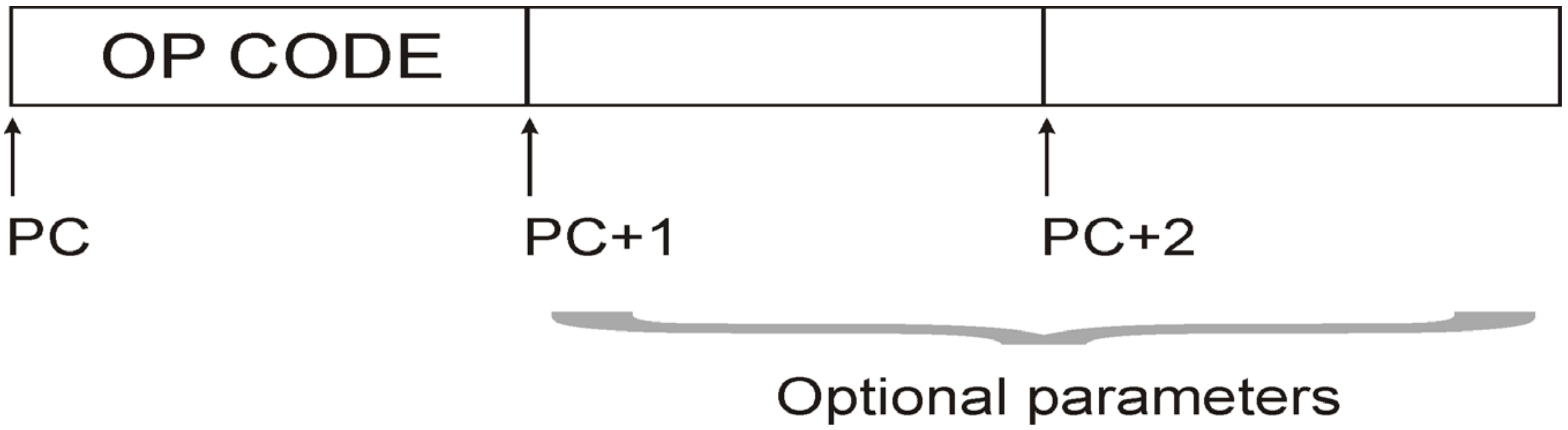
A memory address can be the address of an instruction or data.

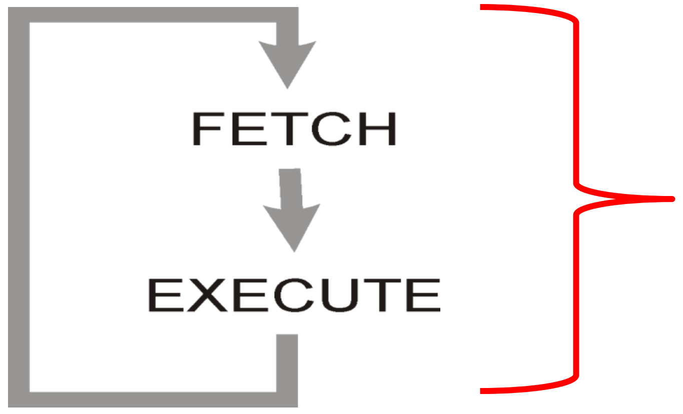
A dedicated memory address register is thus required, to hold an address which can come from several sources.

1. Program Counter will contain the address of the next instruction.

2. Instruction Register may contain the address of data.

3. Instruction Register may contain an address to jump to.





Micro-instruction 1
Micro-instruction 2
Micro-instruction 3
Micro-instruction 4

⋮

Micro-instruction n

- Program is stored in sequential locations in memory
- Program Counter holds the address of the next instruction to fetch
- Copy contents of Program Counter to Address Register
- Read next instruction
- Instruction is copied into instruction register
- Instruction is decoded to produce a sequence of control words

(PC) $\rightarrow$ AR

next instruction $\rightarrow$ IR

$$\left\{\begin{array}{l}\text{(Decode)}\\ \text{Execute sequence of control words}\end{array}\right.$$

(PC) = (PC) + 1

...

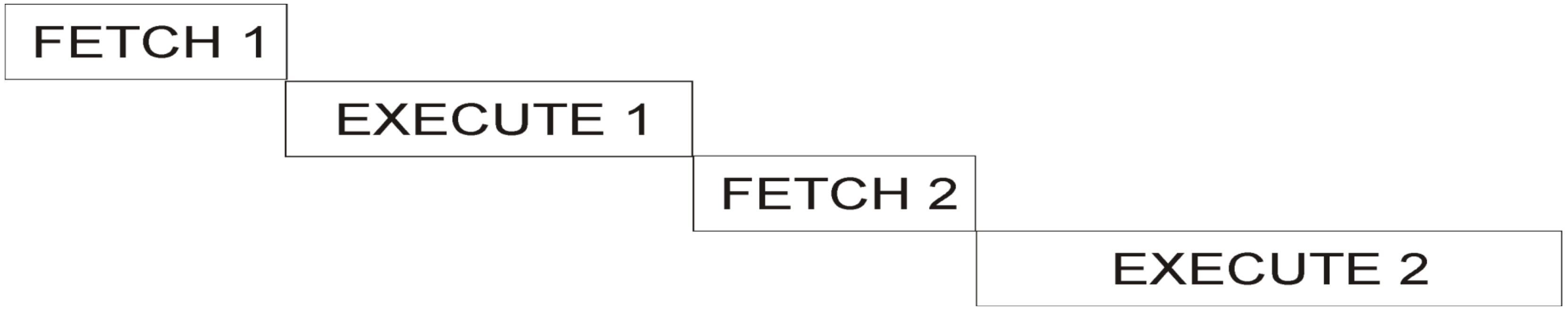
(PC) $\rightarrow$ AR

next instruction $\rightarrow$ IR, (PC) = (PC) + 1

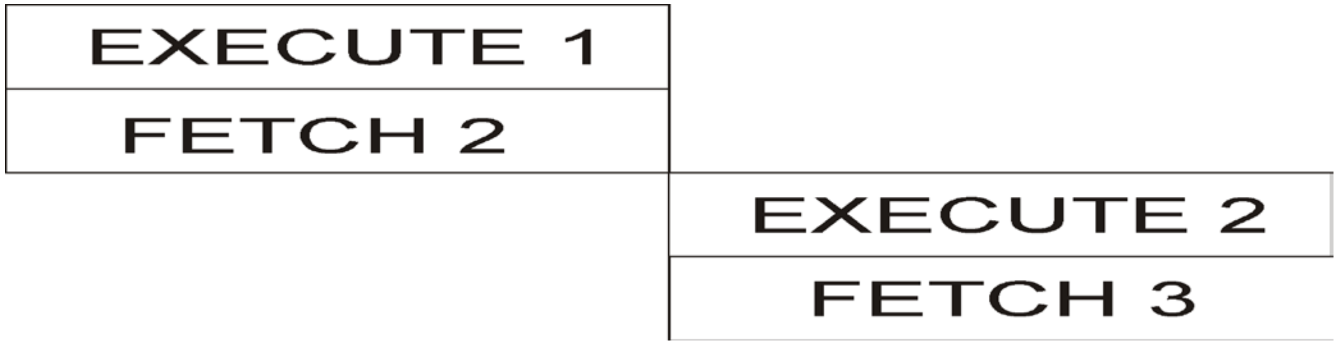
$$\left\{\begin{array}{l}\text{(Decode)}\\ \text{Execute sequence of control words}\end{array}\right.$$

...

Fetch Execute Timeline



Overlapped Fetch Execute



Instruction Cycle

Assume a hardware reset has put us in an initial state, say State A.

We need to reset the Program Counter to point to the first instruction in our program.

We then need to put the value of the Program Counter into the memory Address Register (AR). We can also increment the Program Counter here.

Set up a memory read and transfer the contents of the data bus to the Instruction Register.

Instruction Fetch

State A	PC $\leftarrow$ 0, go to state B
State B	AR $\leftarrow$ PC, PC $\leftarrow$ PC + 1, go to state C
State C	$R/\overline{W} \leftarrow 1$, IR $\leftarrow$ data bus, go to state D
State D	begin the execution phase

Instruction Execution

The first step in the execution phase is to examine the op-code. Consider a two bit op-code (four possible operations)

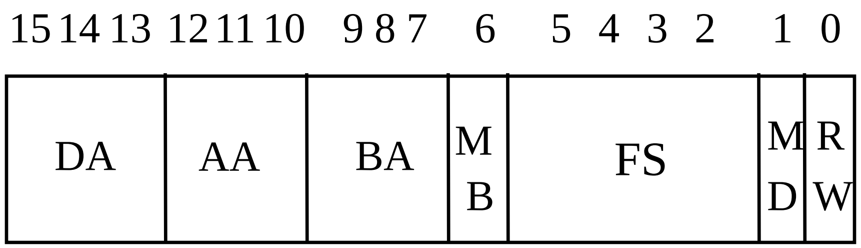
State D   if op-code = 00, then state E
           if op-code = 01, then state F
           if op-code = 10, then state G
           if op-code = 11, then state H

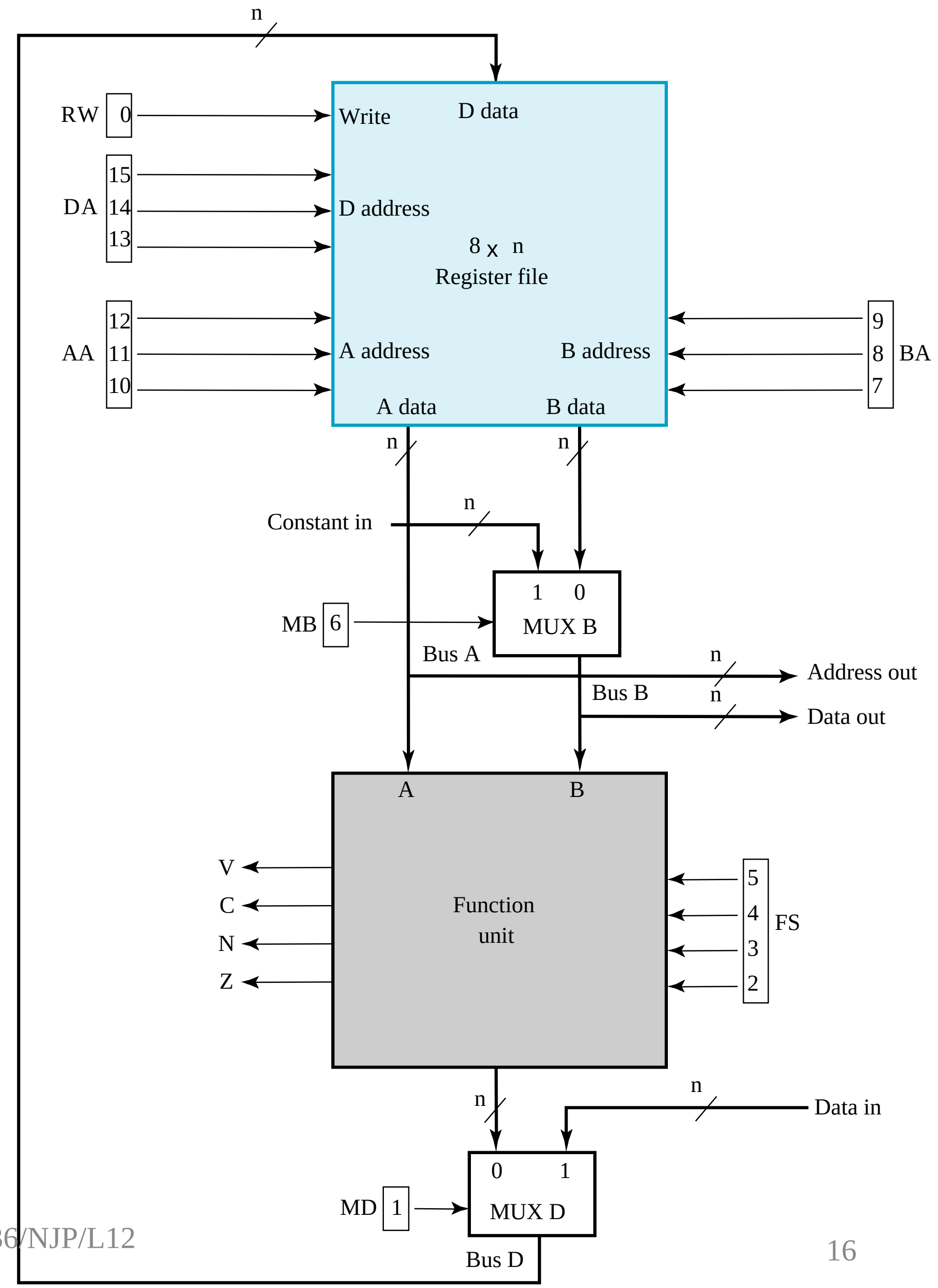
Each of the states will detail the micro-instructions required to execute the corresponding op-code.

The Control Word

- The datapath has many control inputs.
- The signals driving these inputs can be defined and organized into a control word.
- To execute a microinstruction, we apply control word values for a clock cycle. For most micro-operations, the positive edge of the clock cycle is needed to perform the register load.

Example Control word





Fields

- DA – D Address
- AA – A Address
- BA – B Address
- MB – Mux B
- FS – Function Select
- MD – Mux D
- RW – Register Write

Control Unit Implementation

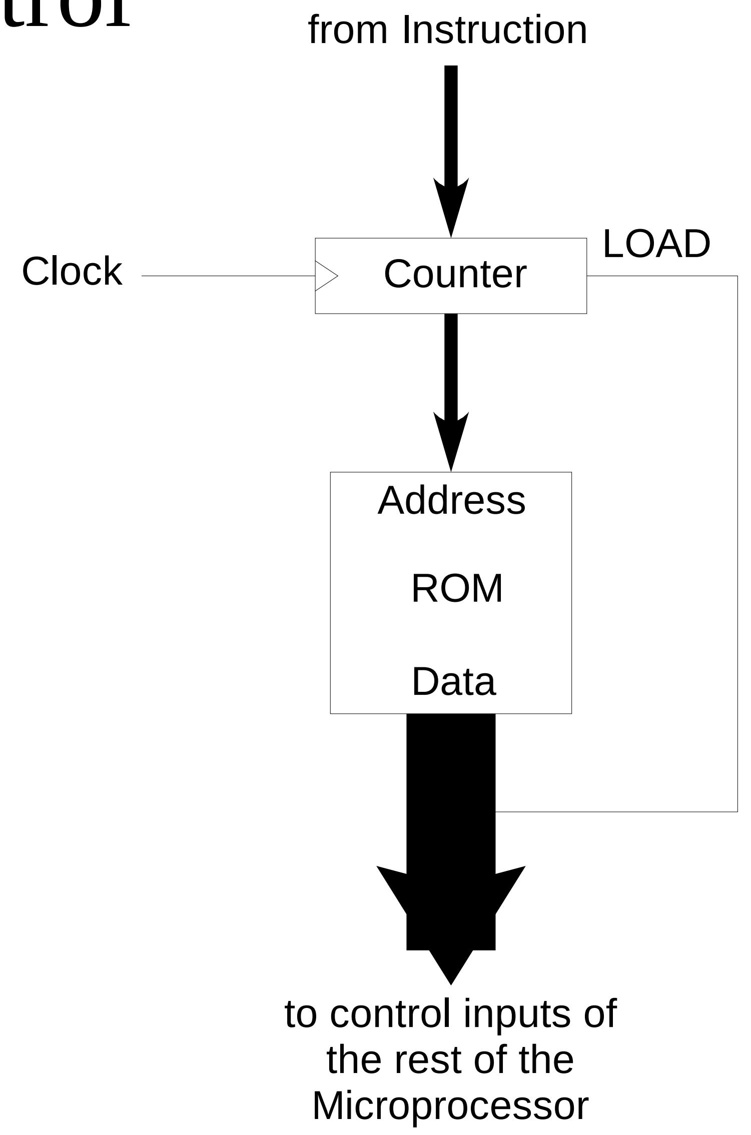
The control unit is a finite state machine in which each state represents a micro-operation. There are two common implementations:

Hard-wired : finite state machine consisting of flip-flops and combinatorial logic.
Can only be modified by a change in hardware.

Micro-programmed : Uses ROM and a micro-program sequencer.
The ROM stores the control signals and the sequencer selects the required micro-operation from the ROM

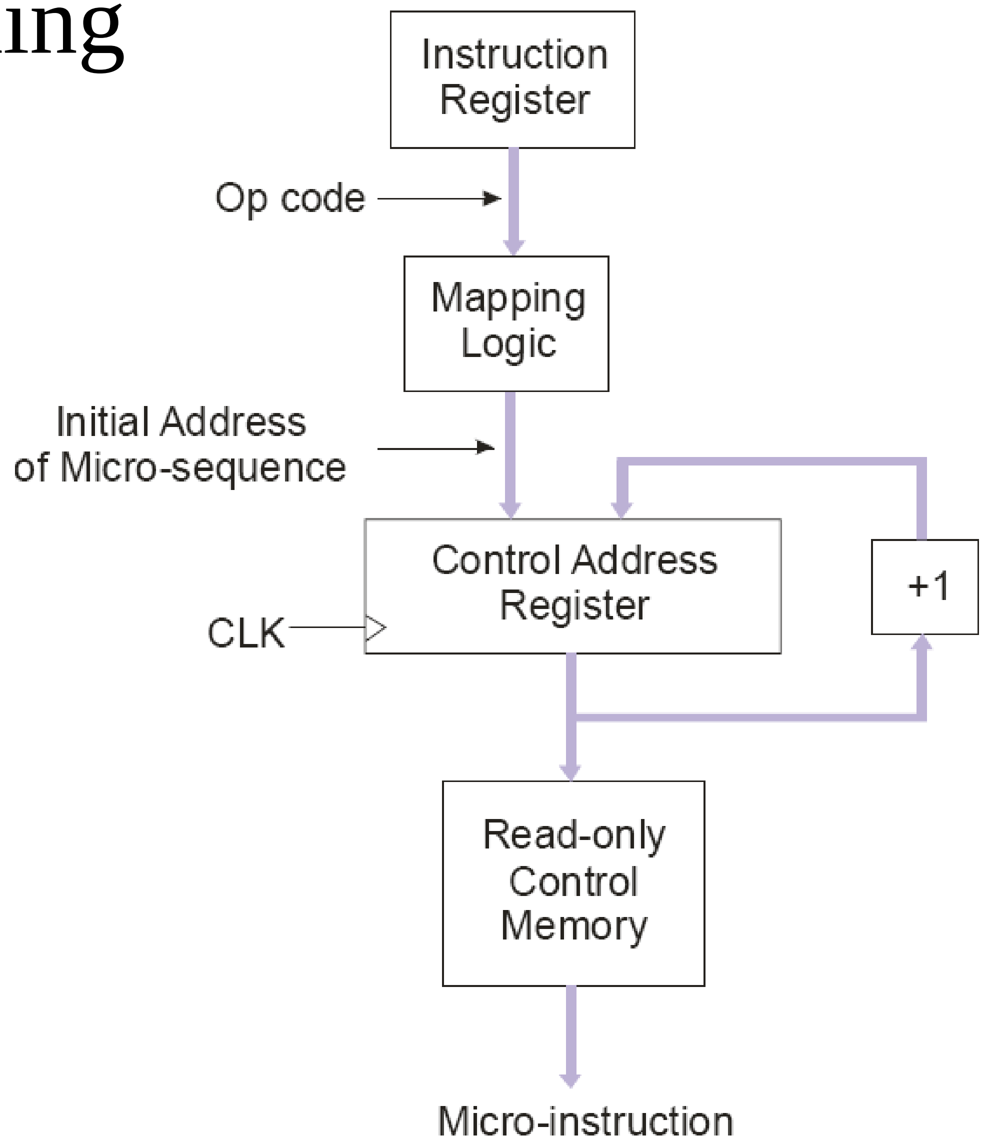
Micro-Programmed Control

- ROM holds sequences of microcode instructions
- Instruction is used to load the counter with the start address of the relevant sequence
- Each bit of the ROMs output is connected to a control input somewhere in the μP
- waggling the control lines up and down makes the μP do something useful



Simplistic Microcoding

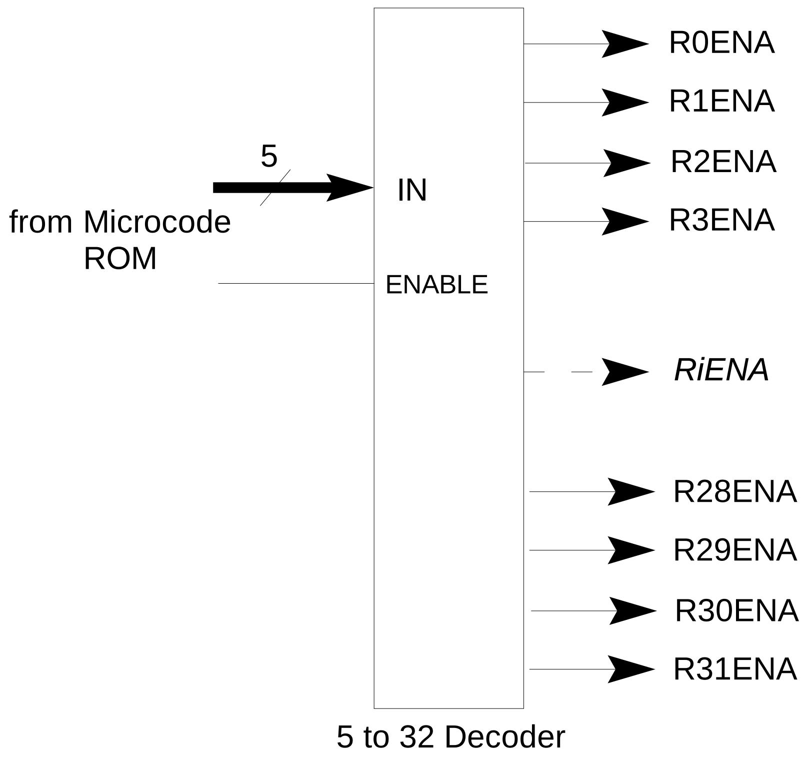
- Op-code selects start of sequence to be generated.
- Sequence stored in contiguous ROM locations.
- Address incremented at each clock cycle to produce the required sequence

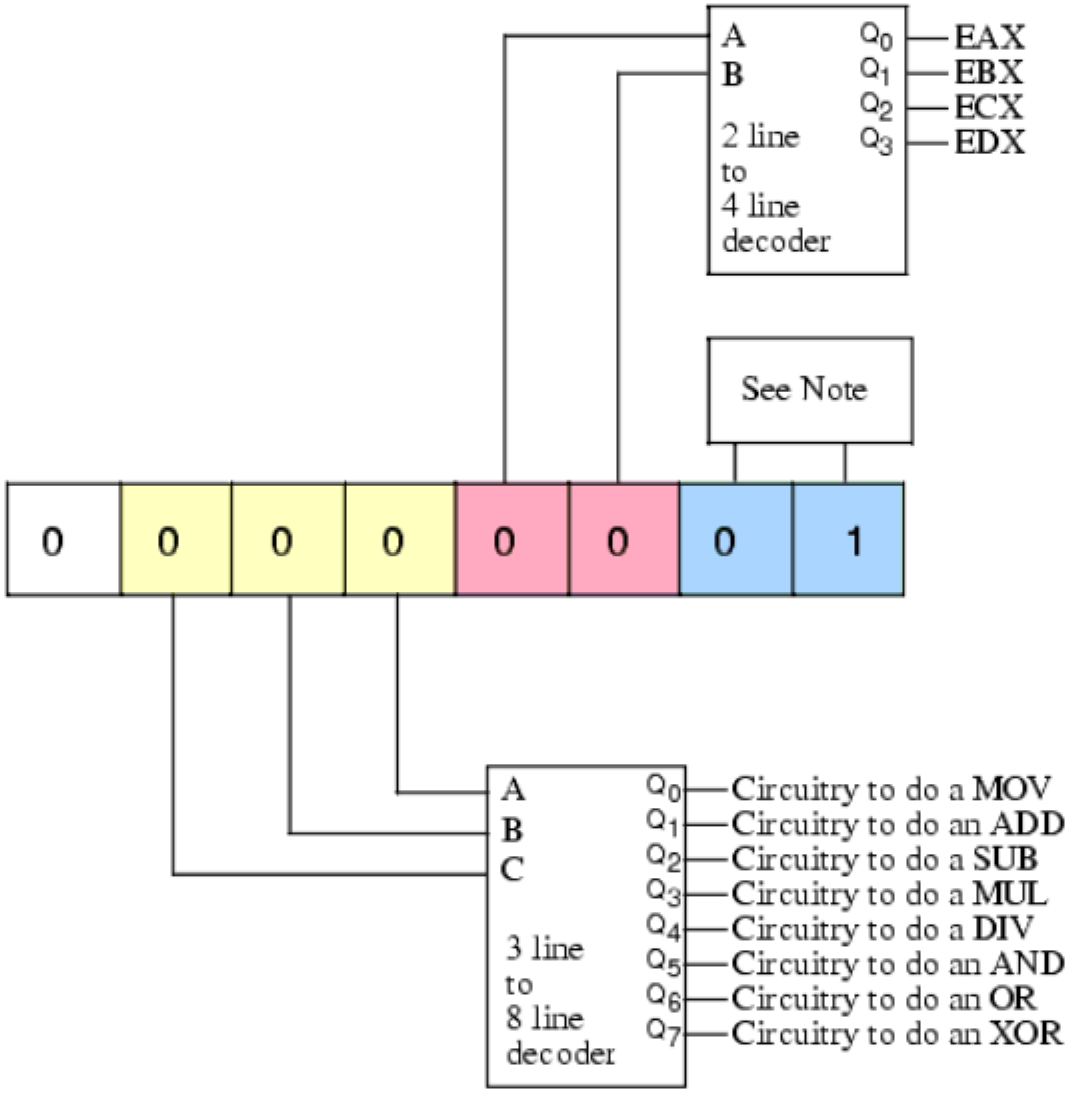


Vertical and Horizontal Microcoding

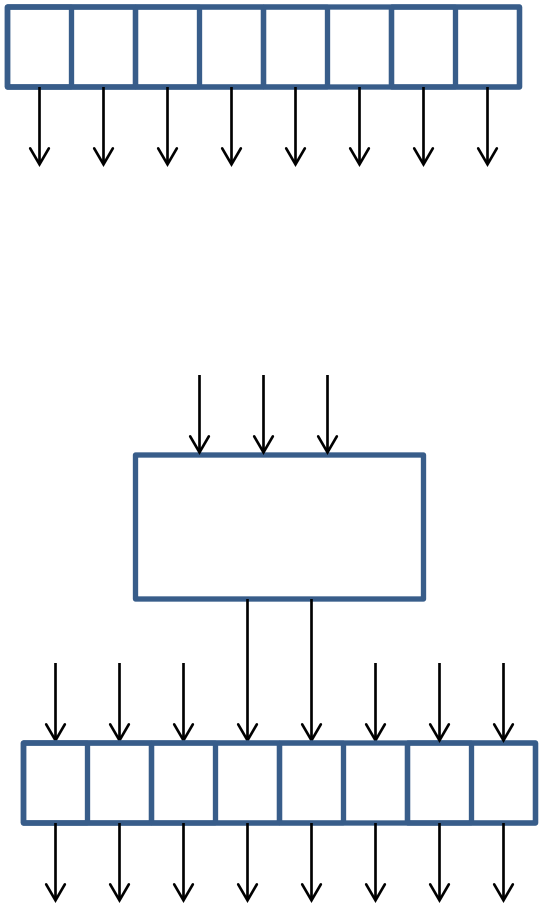
Consider General Register Architecture with 32 registers.

- To enable one (or none) or 32 registers on to bus A would require 32 outputs from the ROM - this is horizontal i.e. wide words
- If decoder is inserted the number of outputs drops to 6 - this is vertical i.e. narrower words





A
B
2 line to 4 line decoder
Q0 EAX
Q1 EBX
Q2 ECX
Q3 EDX
See Note
0 0 0 0 0 0 0 1
A
B
C
3 line to 8 line decoder
Q0 Circuitry to do a MOV
Q1 Circuitry to do an ADD
Q2 Circuitry to do a SUB
Q3 Circuitry to do a MUL
Q4 Circuitry to do a DIV
Q5 Circuitry to do an AND
Q6 Circuitry to do an OR
Q7 Circuitry to do an XOR



Microcoding vs Hard Wired

Microcoding is:

- Better suited to implementing complex control paths
- Easier to design and modify
- Slower than hard wired
- Can be inefficient if complex instructions not used a lot

Principle of Locality

- This is really an observation, not a principle.

- Temporal Locality.
  - If a memory location has been accessed, it will *tend* to be accessed again soon

- Spatial Locality.
  - If a given memory location has been accessed, the next location to be accessed will *tend* to be nearby

There are two distinct classes of architectures:

- Reduced Instruction Set Computers (RISC)
- Complex Instruction Set Computers (CISC)

Before RISC, architectures tried to match closely the operations used in programming languages to make programs compact and use less memory.

Instruction sets were designed to support high-level programming constructs e.g. procedure calls, complex addressing, loop control

Instructions became more complex and this was supported by advances in technology.

CISC Properties

1. Memory access is directly available to most types of instruction.
2. Addressing modes are substantial in number.
3. Instruction formats are of different lengths.
4. Instructions perform both elementary and complex operations.

It is likely that micro-programmed control will be necessary because of the complexity of the instructions.

Program Characteristics

In the early 1980s researchers started to examine the way high level languages like ‘C’ were mapped onto CPUs. The following observations were made.

- The most common instructions were register-to-register moves
- The most frequently accessed data were scalars
- The principle of locality holds for data
- Procedure call and return are costly

RISC Properties

Provide lots of Registers! Some RISC machines have up to 528 registers although 32 to 128 is more common.

- Limit memory accesses to load and store instructions
- Data manipulation instructions are register to register
- Keep most commonly used scalars in registers
- Make procedure calling faster by using registers for parameter passing

RISC Properties

- Multi-stage instruction pipeline
  - Fetch-decode-execute as distinct stages
  - Instruction prefetch
  - Cache memory accelerators

- Fixed instruction width (typically 32-bits)
  - Makes the implementation of multi-stage pipelines easier, no additional operand fetching

CISC vs RISC

- Are RISC processors faster than CISC processors?
  - Definitive tests are hard to make since CPUs are not produced in CISC and RISC versions
  - Compiler implementation differences make comparison difficult
  - Many RISC processors now include CISC features
  - Many CISC processors have borrowed many RISC ideas such as fixed instruction widths and optimised instruction pipelines

Digital Signal Processors (DSP)

DSP covers activities such as:

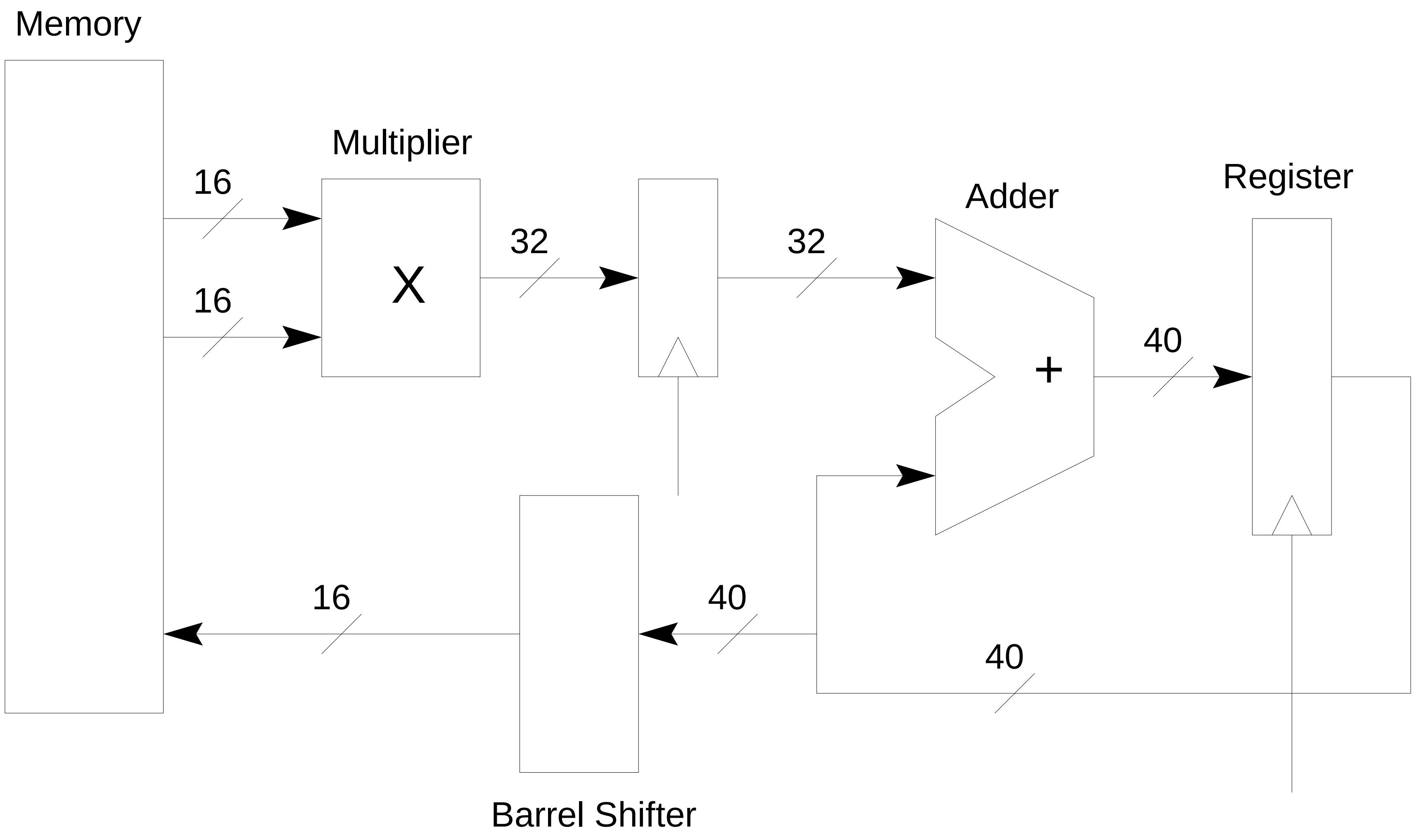
- filtering
- signal transformation (e.g. Fourier Transform)
- signal identification

Implementation of such algorithms have many common attributes:

- Algorithmically similar
- Numerically intensive
- High precision often required
- Real-time
- Interface to analogue I/O

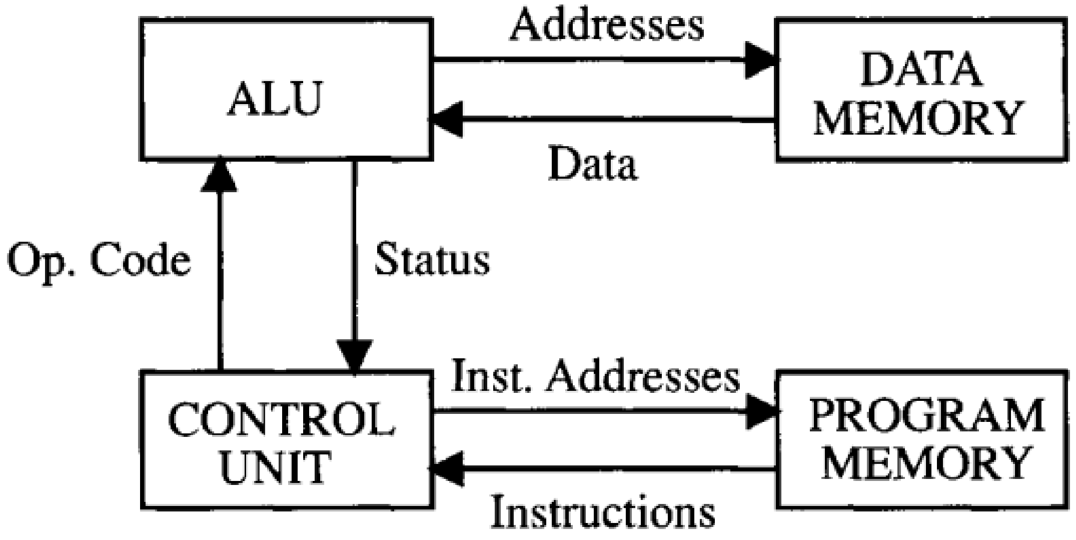
IC manufacturers have developed families of Digital Signal Processors

A Digital Signal Processor must perform multiply/accumulate operations quickly. Implementation:



Memory Interface

- Digital Signal Processors often physically separate their program and data memory spaces, accessing each through separate, physical address, data, and control buses.
- This architecture which is called ‘Harvard’ allows program memory accesses to run in parallel with data memory accesses and increases performance.


ALU
Addresses
DATA MEMORY
Data
Op. Code
Status
Inst. Addresses
CONTROL UNIT
PROGRAM MEMORY
Instructions