OLLSCOIL TEICNEOLAÍOCHTA BHAILE ÁTHA CLIATH

DUBLIN

TECHNOLOGICAL UNIVERSITY DUBLIN

Semester 1, Week 9

Scoil na Ríomheolaíochta
School of Computer Science

TU856-1 & TU858-1 Computer Architecture and Technology

Module Code: CMPU 1006

VON NEUMANN ARCHITECTURE

Presentation Outline

This presentation begins with a historic reference to the person credited with computer architecture as it has become known.

It will focus on The electronic components that need the architecture, and they are investigated for their functionality.

There is a number of examples of the fetch-execute cycle associated with von Neumann architecture.

Presentation Content - including

John von Neumann

“The father of the modern computer.”

- This quote is often cited in relation to John von Neumann.
- Born in Hungary and named Neumann János Lajos Margittai, John spent a lot of time as a professor at Princeton University, USA.
- A lot of his work was questionable (atomic bomb development) but he was a scientific genius. (Physics, mathematics, chemistry…)

John von Neumann (2)

- Mauchly and Eckert (as mentioned in the notes of Week 1) had mastered the architecture advances for which von Neumann has become famous while developing the ENIAC computer.

- Von Neumann worked with Mauchly and Eckert and helped with the ENIAC project – but it was his publication of a journal paper on the internal workings of an ENIAC-type machine (called ‘First Draft’) that earned him the accolades.

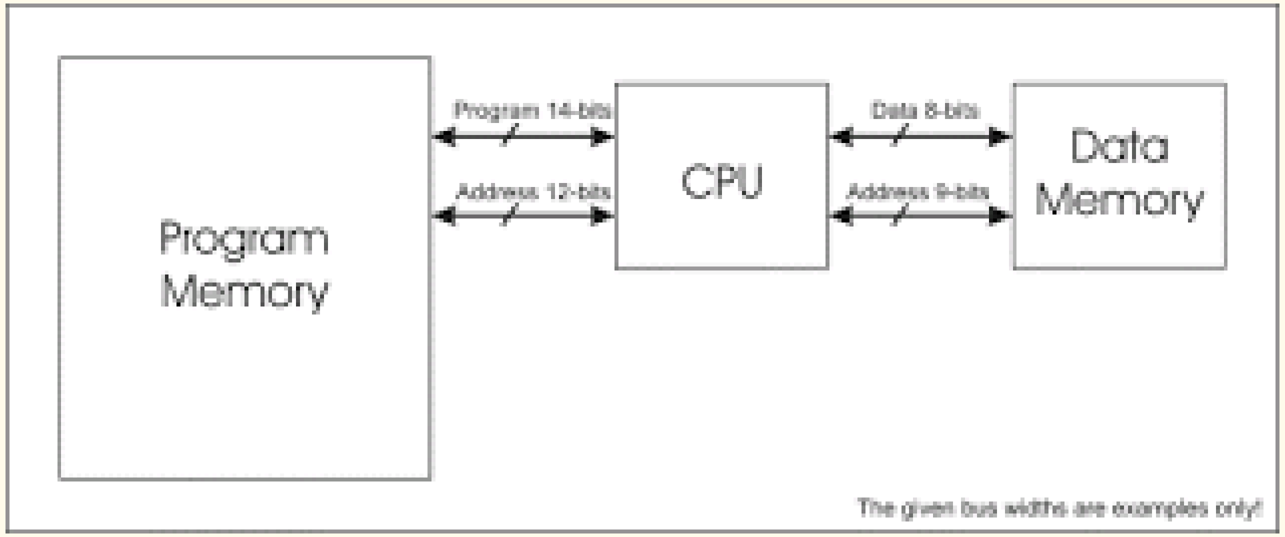
The Architecture

- Before the ENIAC (Electronic Numerical Integrator And Computer) the electrical computing machines were manually loaded with instructions that were executed as they came through the processor – causing immediate output.
- Von Neumann Architecture is based on ‘the stored program concept’.
- Going from ‘in-the-moment’ computing to stored programs was a big leap in computing, at the time (mid-1940s).

The Architecture (2)

- The von Neumann Architecture is a computer design model that uses a single storage structure to hold both instructions and data.
- A computer which implements the ‘referential model’ of specifying sequential architectures, in contrast with parallel architectures is described as having von Neumann Architecture.
- The separation of storage from the processing unit is implicit in the von Neumann Architecture.

Referential Model



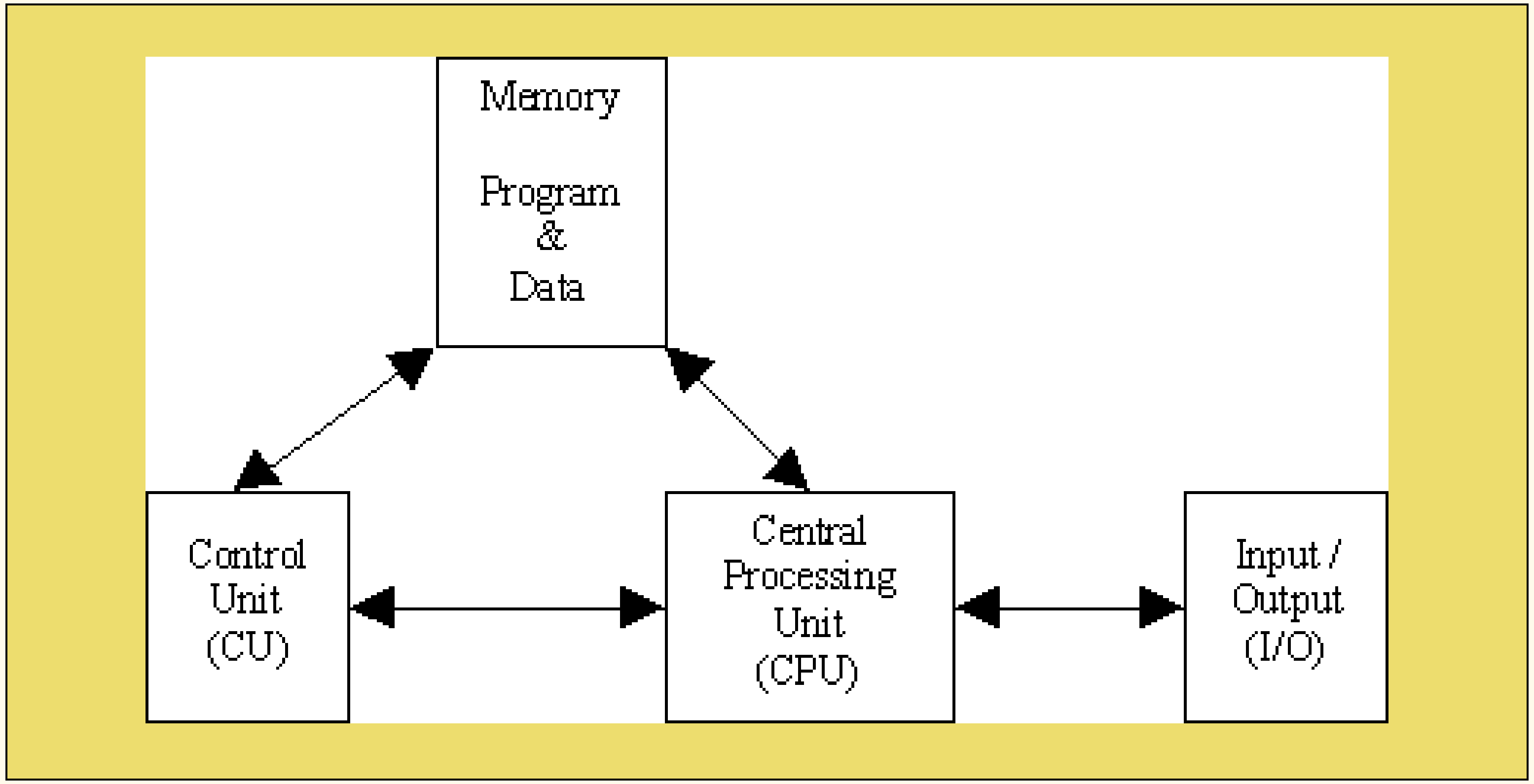
The architecture developed by von Neumann, Mauchly and Eckert (as above), and the 'stored program concept' are the basis of the Referential Model.

The Architecture (3)

- The term ‘stored-program computer’ is generally used to mean a computer of this design.
- Von Neumann begins his idea with a broad description of the general-purpose computing machine containing four main sub-components.
- The sub-components relate to memory, control, central processing and connection with the human operator (sometimes defined as input/output).
- These are represented by the arithmetic logic unit, the control unit, the memory, and the input-output devices of the classical computer model.

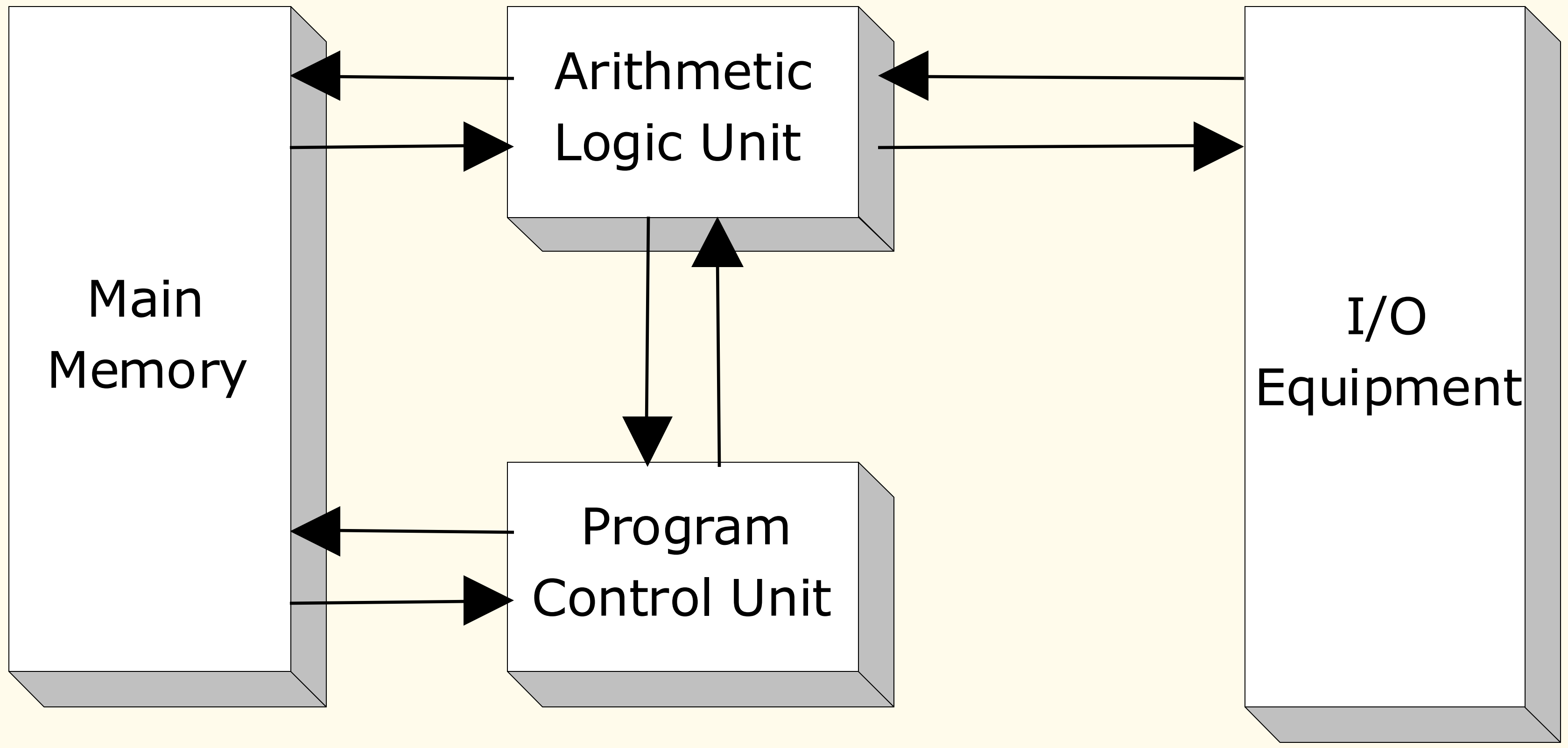
The Architecture (4)

Diagram of the Architecture:



The Architecture (5)

Another diagram of the Architecture – this matches the ENIAC architecture:



The Architecture (6)

- In a special purpose machine the computational procedure could be part of the hardware. In a general purpose one the instructions must be as changeable as the numbers they acted upon.
- To von Neumann the key to building a general purpose computing machine was in its ability to store not only its data and the intermediate results of computation, but also to store the instructions, or orders that allowed the computation.

The Architecture (7)

- Therefore, why not encode the instructions into numeric form and store instructions and data in the same memory?
- This frequently is viewed as the principal contribution provided by von Neumann's insight (and Mauchly and Eckert’s) into the nature of what a computer should be.

The Architecture (8)

- The stored program concept was proposed about sixty-five years ago – and it is the <u>fundamental</u> architecture employed by computers, even now.

- Computer technologies come and go (or get superseded) but the stored-program concept endures.

The Stored Program Concept

The stored program concept had several technical implications:

- Four key sub-components operate together to make the stored program concept work.
- The process that moves information through the sub-components is called the ‘Fetch-Execute Cycle’.
- Unless otherwise indicated, program instructions are executed in sequential order.

The Four Sub-Components

Physically, the four sub-components in von Neumann Architecture might be called:

- Input/Output (or ‘I/O’)
- Control Unit
- Arithmetic-Logic Unit
- Memory

While only four sub-components are listed in the books, there is a fifth part to the operation: a bus (or wire) that connects the components together and over which data flow from one sub-component to another. (More on buses next week.)

Memory

There are several different types of memory at processor level:

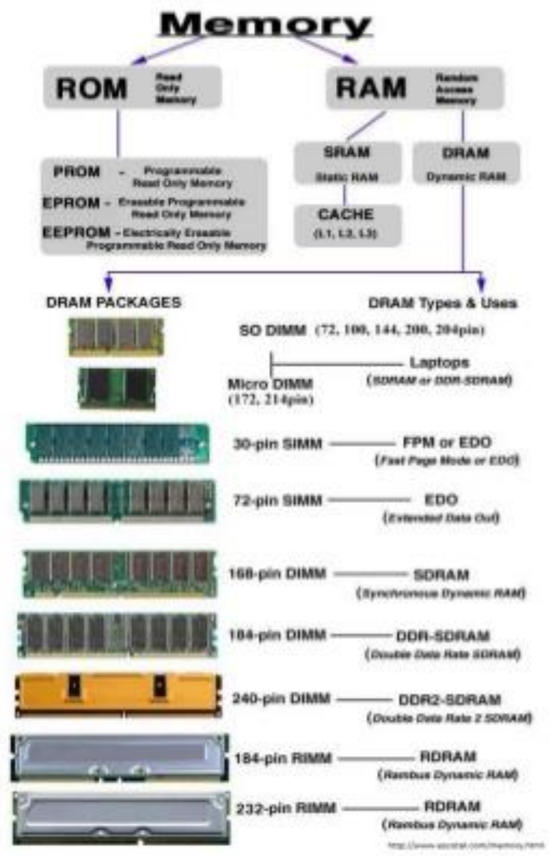
- RAM (Random Access Memory)
- ROM (Read Only Memory)
- Registers
- Cache

- Why not just engineer one type?
- Each type of memory represents cost/benefit tradeoffs between capability and cost …

Memory (2)

“Memory”

- RAM
- ROM
- Flash memory (EPROM)
- Cache (L1, L2)
- Dynamic RAM (DRAM)
- Static RAM (SRAM)
- ThumbDrives
- Virtual Memory
- Video Memory
- BIOS
- Hard Disk – Permanent Memory


Memory
ROM Read Only Memory
RAM Random Access Memory
PROM - Programmable Read Only Memory
EPROM - Erasable Programmable Read Only Memory
EEPROM - Electrically Erasable Programmable Read Only Memory
SRAM Static RAM
DRAM Dynamic RAM
CACHE (L1, L2, L3)
DRAM PACKAGES
DRAM Types & Uses
SO DIMM (72, 100, 144, 200, 204pin)
Micro DIMM (172, 214pin)
Laptops (SDRAM or DDR-SDRAM)
30-pin SIMM — FPM or EDO (Fast Page Mode or EDO)
72-pin SIMM — EDO (Extended Data Out)
168-pin DIMM — SDRAM (Synchronous Dynamic RAM)
184-pin DIMM — DDR-SDRAM (Double Data Rate SDRAM)
240-pin DIMM — DDR2-SDRAM (Double Data Rate 2 SDRAM)
184-pin RIMM — RDRAM (Rambus Dynamic RAM)
232-pin RIMM — RDRAM (Rambus Dynamic RAM)

Memory Types

- RAM is an array of cells, each with a unique address.
- It should take the same amount of time to access any memory cell, regardless of its location with the memory bank. (‘Random’ access.)

- ROM memory cells can be read from, but not written to. Cells are protected.
- ROM is more expensive than RAM and only critical information is stored. (I.E. Firmware instructions.)

Memory Types (2)

- Register cells are powerful, costly, and physically located close to the heart of computing.
- Some registers are the main participants in the Fetch-Execute Cycle. (More on the Fetch-Execute Cycle later.)

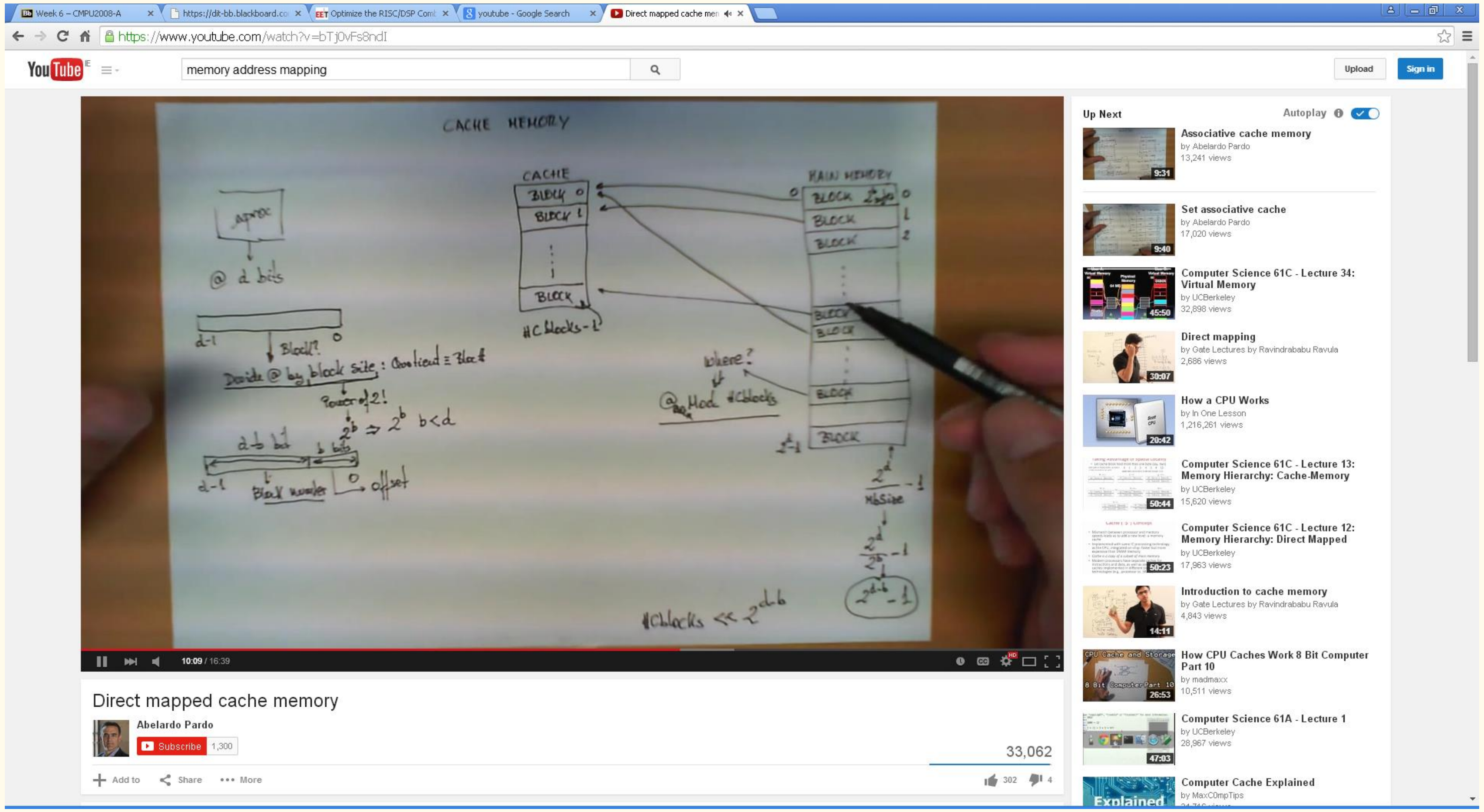
- Cache memory is a type of RAM. The main difference between cache and actual RAM is that cache is addressed by the CPU before RAM - so instructions may be performed quicker if in cache.

Memory Addressing

- For all these memory types, cell size or cell width is a measure of how many individual memory cells (switches for bits) are addressed at a time.

- At a minimum, this is a byte (8 bits) but to support all data types and operations cell size can be larger (example, a word of 16 bits).

Memory Addressing (2)



Direct mapped cache memory

Abelardo Pardo

Subscribe 1,300

33,062

Add to Share More 302 4

Up Next Autoplay

Associative cache memory	by Abelardo Pardo	13,241 views
Set associative cache	by Abelardo Pardo	17,020 views
Computer Science 61C - Lecture 34: Virtual Memory	by UCBerkeley	32,898 views
Direct mapping	by Gate Lectures by Ravindrababu Ravula	2,686 views
How a CPU Works	by In One Lesson	1,216,261 views
Computer Science 61C - Lecture 13: Memory Hierarchy: Cache-Memory	by UCBerkeley	15,620 views
Computer Science 61C - Lecture 12: Memory Hierarchy: Direct Mapped	by UCBerkeley	17,963 views
Introduction to cache memory	by Gate Lectures by Ravindrababu Ravula	4,843 views
How CPU Caches Work 8 Bit Computer Part 10	by madmaxx	10,511 views
Computer Science 61A - Lecture 1	by UCBerkeley	28,967 views
Computer Cache Explained	by MaxCOmpTips	

https://www.youtube.com/watch?v=bTj0vFs8ndI

Memory Addressing (3)

- All cells have an address and can contain data (where data = data or instructions).
- The cell address is a label (like a post code) that identifies a particular cell.
- The cell contents are whatever data are stored at a given address location.

Memory Addressing (4)

- The memory labels are themselves binary numbers.
- One of the internal registers is used to hold address locations. (MAR – memory address register or machine address register.)
- This register has to be big enough to hold the biggest address of a memory location.

I/O: Input and Output

- There is both a human-machine interface and a machine-machine interface to I/O.
- Examples of the human-machine interface include a keyboard, screen or printer.
- Examples of the machine-machine interface include things like mass storage and secondary storage devices.

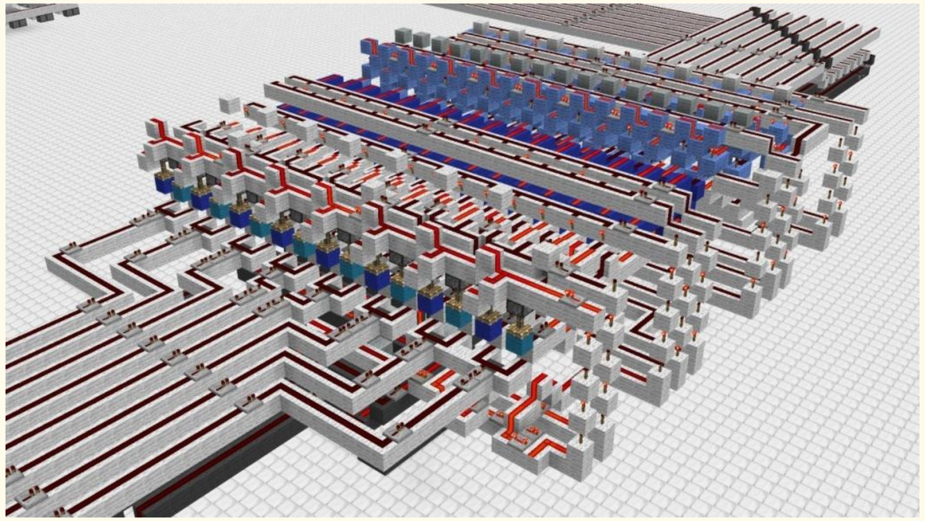
Arithmetic Logic Unit - The ALU

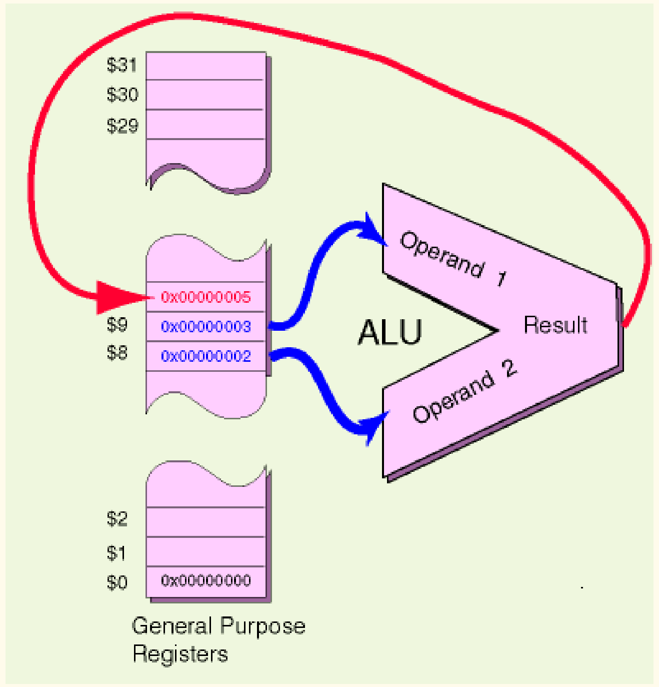
- The third component in the von Neumann Architecture is called the Arithmetic Logic Unit.

- This is the subcomponent that performs the arithmetic and logic operations based on combinations of flip-flops, latches and gates, generally.

- The ALU is the “brain” of the computer.

The ALU (2)





The graphic on the left is a 3-D lego-type representation of the ALU

The ALU (3)

- The ALU contains many of the registers.

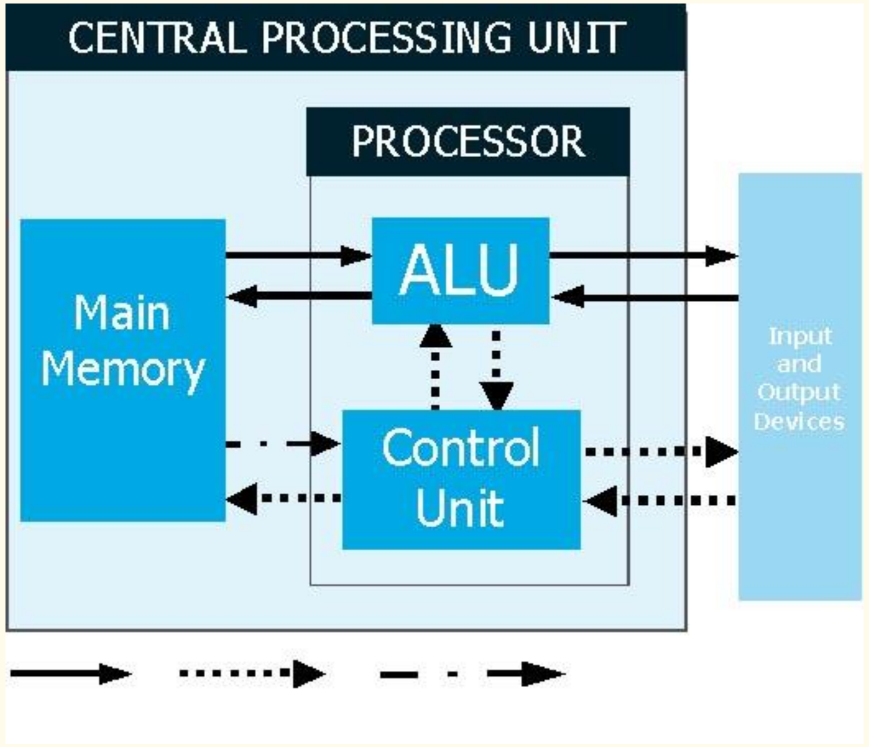
- Its flip-flop and latch circuitry allows it to perform addition, subtraction, multiplication and division, as well as logical comparisons: less than, equal to and greater than.

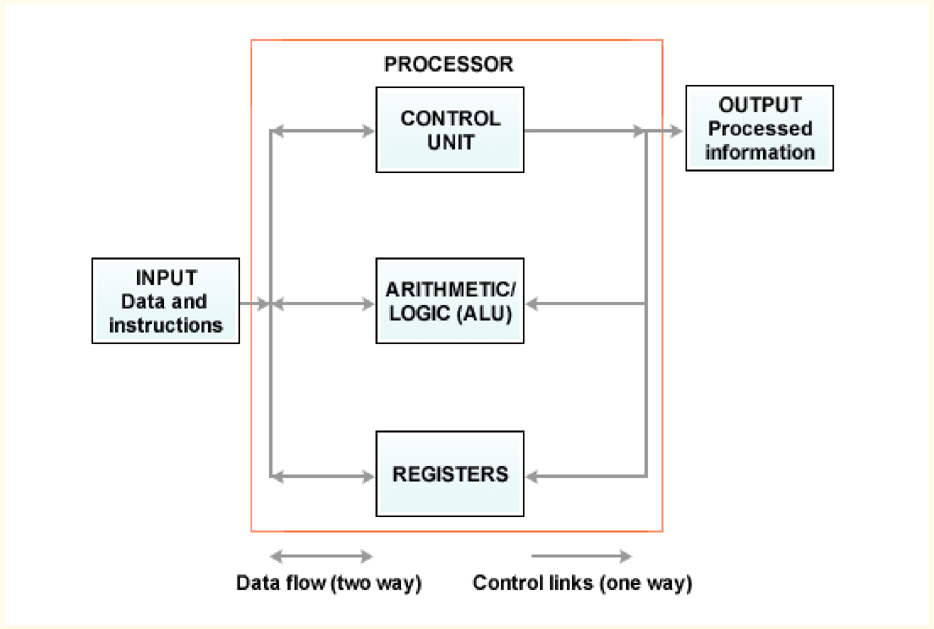
Control Unit

- The control unit drives the fetch and execute cycle.

- As mentioned before, in memory, a cell address is loaded into the MAR – it is the control unit that figures out which address is loaded, and what operation is to be performed with the data moved to the memory data register (or machine data register) - MDR.

Control Unit (2)





Control Unit (3)

- To do its job, the Control Unit has several tools, including;
    - Special memory registers
    - ‘Hard-wired’ understanding of an Instruction Set

Control Unit (4)

Special Memory Registers

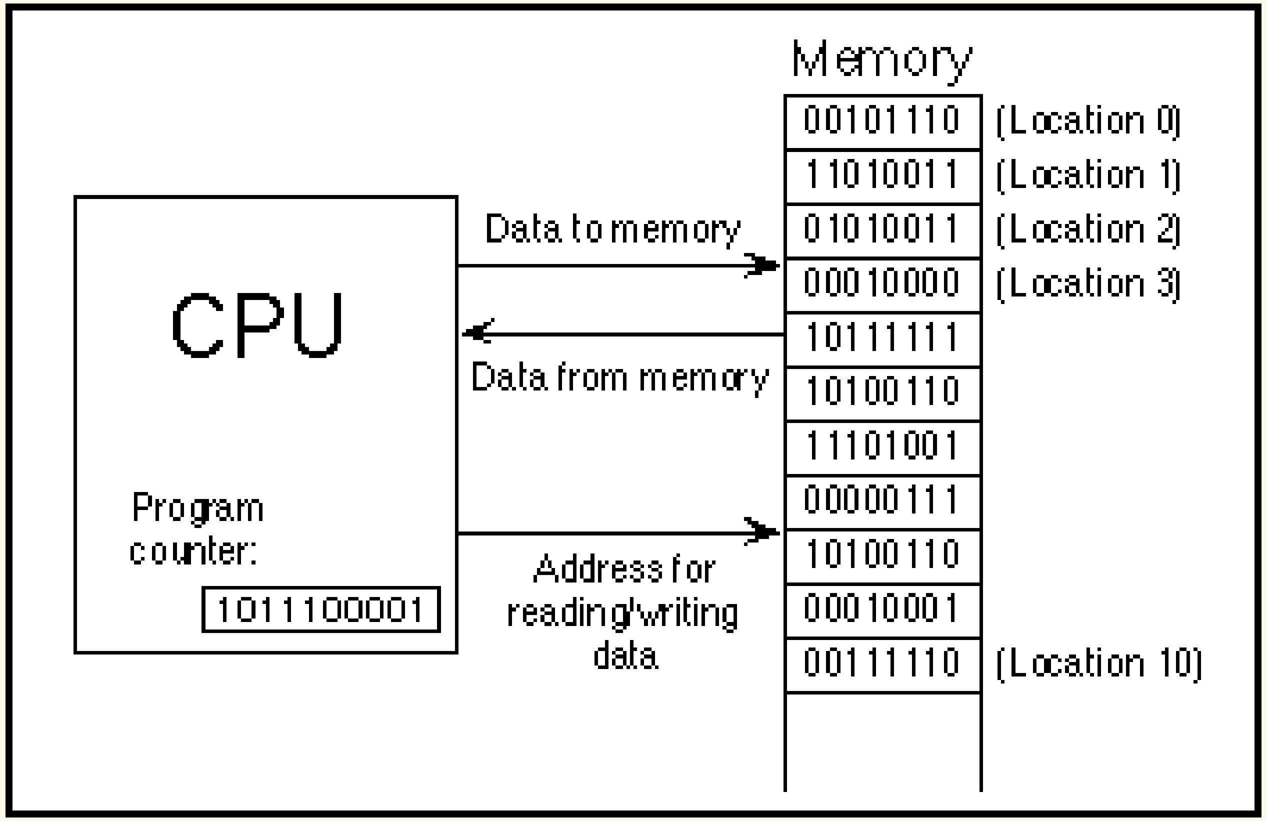
- The Control Unit must keep track of where it is within a program, and what it should do next.
- Two special registers are used to accomplish this:
    - A program counter, typically referred to as a PC, holds the address of the next instruction to be executed.
    - An instruction register, typically referred to as an IR, holds an instruction fetched from memory.

Control Unit (5)

- Along with the special registers, the Control Unit utilises special circuitry, called an instruction decoder.

- The instruction decoder is a typical decoder circuit, and its purpose is to read an instruction from the instruction register (IR), and then to activate the appropriate circuit line.

Simple View of CPU in Operation



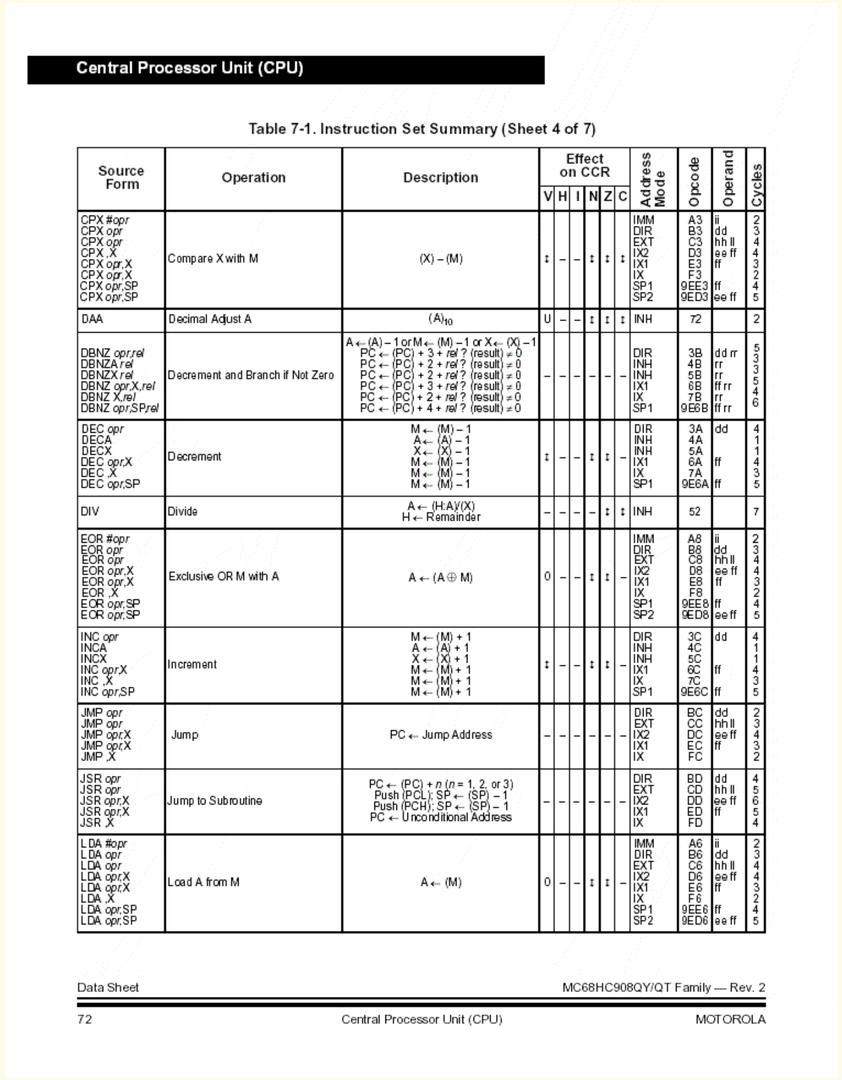
How this Works

- Remember, we are trying to figure out how the stored program concept works.
- In this model, the program and the data upon which it operates are stored in memory locations.
- We know how to encode the data.
- We need to learn how to encode the programming instructions.

The Instruction Set

- At the heart of all programming are a few basic, general instructions.
- The set of instructions is small and particular to the processor’s specification.
- The power of the instruction set is that by identifying a definite, bounded, simple task, an instruction can be executed with appreciable speed (typically within a few billionths of a second).

The Instruction Set (2)



Central Processor Unit (CPU)

Table 7-1. Instruction Set Summary (Sheet 4 of 7)

Source Form	Operation	Description	Effect on CCR						Address Mode	Opcode	Operand	Cycles
			V	H	I	N	Z	C				
CPX #*opr* CPX *opr* CPX *opr* CPX ,X CPX *opr*,X CPX *opr*,X CPX *opr*,SP CPX *opr*,SP	Compare X with M	(X) – (M)	↕	–	–	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP1 SP2	A3 B3 C3 D3 E3 F3 9EE3 9ED3	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
DAA	Decimal Adjust A	$(A)_{10}$	U	–	–	↕	↕	↕	INH	72		2
DBNZ *opr,rel* DBNZA *rel* DBNZX *rel* DBNZ *opr*,X,*rel* DBNZ X,*rel* DBNZ *opr*,SP,*rel*	Decrement and Branch if Not Zero	A ← (A) – 1 or M ← (M) – 1 or X ← (X) – 1 PC ← (PC) + 3 + *rel* ? (result) ≠ 0 PC ← (PC) + 2 + *rel* ? (result) ≠ 0 PC ← (PC) + 2 + *rel* ? (result) ≠ 0 PC ← (PC) + 3 + *rel* ? (result) ≠ 0 PC ← (PC) + 2 + *rel* ? (result) ≠ 0 PC ← (PC) + 4 + *rel* ? (result) ≠ 0	–	–	–	–	–	–	DIR INH INH IX1 IX SP1	3B 4B 5B 6B 7B 9E6B	dd rr rr rr ff rr rr ff rr	5 3 3 5 4 6
DEC *opr* DECA DECX DEC *opr*,X DEC ,X DEC *opr*,SP	Decrement	M ← (M) – 1 A ← (A) – 1 X ← (X) – 1 M ← (M) – 1 M ← (M) – 1 M ← (M) – 1	↕	–	–	↕	↕	–	DIR INH INH IX1 IX SP1	3A 4A 5A 6A 7A 9E6A	dd ff ff	4 1 1 4 3 5
DIV	Divide	A ← (H:A)/(X) H ← Remainder	–	–	–	–	↕	↕	INH	52		7
EOR #*opr* EOR *opr* EOR *opr* EOR *opr*,X EOR *opr*,X EOR ,X EOR *opr*,SP EOR *opr*,SP	Exclusive OR M with A	A ← (A ⊕ M)	0	–	–	↕	↕	–	IMM DIR EXT IX2 IX1 IX SP1 SP2	A8 B8 C8 D8 E8 F8 9EE8 9ED8	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
INC *opr* INCA INCX INC *opr*,X INC ,X INC *opr*,SP	Increment	M ← (M) + 1 A ← (A) + 1 X ← (X) + 1 M ← (M) + 1 M ← (M) + 1 M ← (M) + 1	↕	–	–	↕	↕	–	DIR INH INH IX1 IX SP1	3C 4C 5C 6C 7C 9E6C	dd ff ff	4 1 1 4 3 5
JMP *opr* JMP *opr* JMP *opr*,X JMP *opr*,X JMP ,X	Jump	PC ← Jump Address	–	–	–	–	–	–	DIR EXT IX2 IX1 IX	BC CC DC EC FC	dd hh ll ee ff ff	2 3 4 3 2
JSR *opr* JSR *opr* JSR *opr*,X JSR *opr*,X JSR ,X	Jump to Subroutine	PC ← (PC) + *n* (*n* = 1, 2, or 3) Push (PCL); SP ← (SP) – 1 Push (PCH); SP ← (SP) – 1 PC ← Unconditional Address	–	–	–	–	–	–	DIR EXT IX2 IX1 IX	BD CD DD ED FD	dd hh ll ee ff ff	4 5 6 5 4
LDA #*opr* LDA *opr* LDA *opr* LDA *opr*,X LDA *opr*,X LDA ,X LDA *opr*,SP LDA *opr*,SP	Load A from M	A ← (M)	0	–	–	↕	↕	–	IMM DIR EXT IX2 IX1 IX SP1 SP2	A6 B6 C6 D6 E6 F6 9EE6 9ED6	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5

Data Sheet MC68HC908QY/QT Family — Rev. 2

72 Central Processor Unit (CPU) MOTOROLA

The Instruction Set (3)

- The instruction set is something like the ASCII alphabet encoding scheme.
- The specific instructions are given unique binary codes.
- Obviously, the instruction register must be big enough to hold any instruction within the numbered set.

Execution Happens in Cycles

- Data and program instructions are translated into binary form and stored in RAM.
- As the information is needed, it is moved to the high speed, costlier registers where it is processed.
- This process occurs in a cycle: fetch information to the registers, and execute it there, fetch the next information from the registers, and execute it, etcetera.

Engineering Needs

- Once we know on which data we should be working, we know how to build circuitry to perform processing operations. (We can add, subtract, divide and compare).

- How do we figure out what data to be working on, and exactly which operation to perform?

Engineering Needs (2)

- Specifically, this is what we need to be able to do:
- Build a circuit that will allow us to take whatever number is in the Memory Address Register (MAR), and use this number to access a specific memory cell.
- Build a circuit that will allow us to choose which data results should be placed in the Memory Data Register (MDR).
- This magic happens in the Control Unit (also known as the PCU (Program Control Unit).

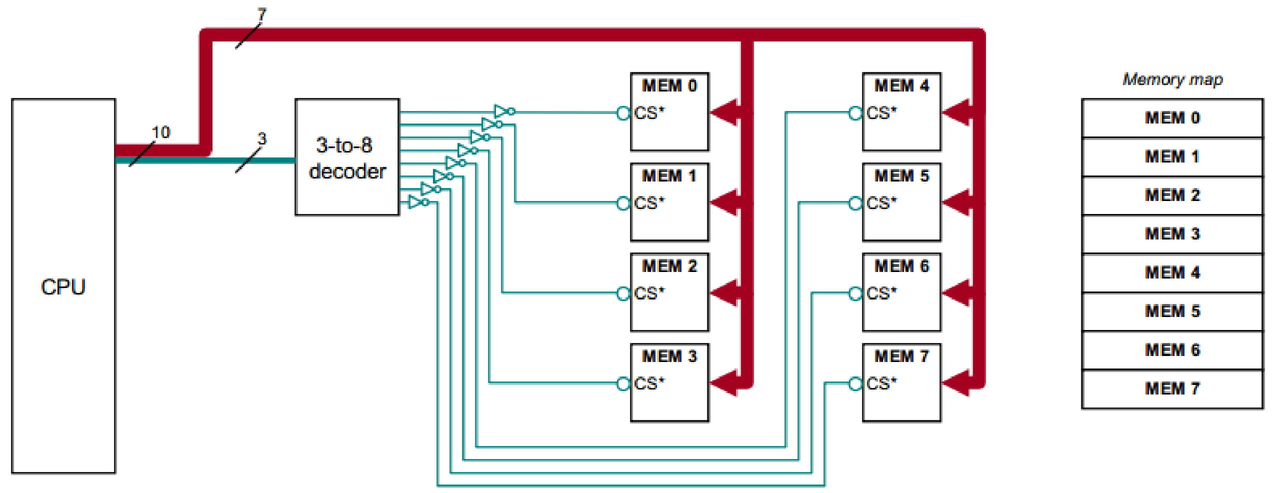
Choosing a Memory Location

- The initial requirement first: how do we determine which address location holds the data on which we need to operate?
- As mentioned before, the MAR holds an address - a binary number.
- We need some circuitry to read that number, and based on its value, find exactly the correct address location to read.
- The circuit is called a decoder …

Decoder

A very simple example

- **Let's assume a very simple microprocessor with 10 address lines (1KB memory)**
- **Let's assume we wish to implement all its memory space and we use 128x8 memory chips**
- **SOLUTION**
  - We will need 8 memory chips (8x128=1024)
  - We will need 3 address lines to select each one of the 8 chips
  - Each chip will need 7 address lines to address its internal memory cells



A von Neumann Architecture Effect

- The idea of the ‘stored program concept’ had a specific effect: The stored program needed to have procedures to allow the individual instructions and data to be ‘called and dealt with’ – a fetch and execute arrangement (or fetch-execute-store arrangement).

The Fetch-Execute Cycle

- The CPU executes a program that is stored as a sequence of machine language instructions in main memory.

- It does this by repeatedly reading, or fetching, an instruction from memory and then carrying out, or executing, that instruction.

The Fetch-Execute Cycle (2)

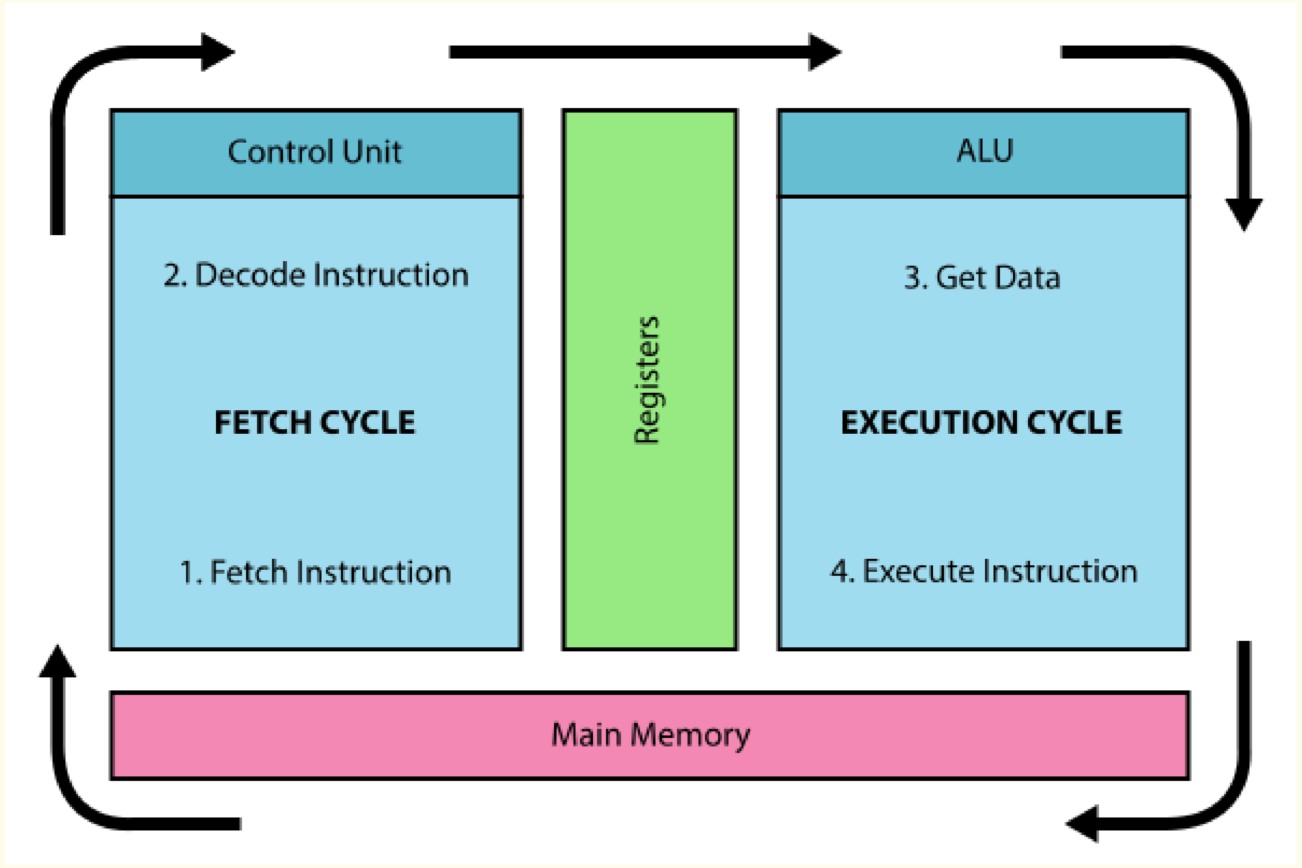
- This process of ‘fetch an instruction, execute it, fetch another instruction, execute it and so on’, is called the Fetch-Execute Cycle.
- This continuous check that the CPU makes; fetching, executing and pushing out the result, is called ‘polling’ but can be interrupted by an ‘interrupt’.

The Fetch-Execute Cycle (3)

This is a two-cycle process because both instructions and data are in memory.

- Fetch
  - Decode or find an instruction, load from memory into a register and signal the ALU.
- Execute
  - This performs an operation that the instruction requires.
  - Move/transform data.

The Fetch-Execute Cycle (4)



The Fetch-Execute Cycle (5)

- With an interrupt the CPU reacts to a signal to ‘jump’ to an address. There will be ‘interrupt handler’ instructions there.
- An interrupt handler is part of the device driver software for the device that signalled the interrupt.
- It will allow the diversion to another program and, after the interrupt, direct the CPU back to the previous program – ‘where it left off…’

The Fetch-Execute Cycle (6)

Instruction cycle example:

The ALU and PCU are used for the fetch-execute cycle:

Arithmetic and Logic Unit - ALU

- The ALU is part of the Central Processing Unit. All computations are performed in this unit.
- The ALU comprises adders, counters, and registers, numerical operations (+ - / x) and logical operations (AND, OR, program branching).

The Fetch-Execute Cycle (7)

(Program) Control Unit - PCU

The PCU :

- Coordinates all other units in the computer,
- Organises movement of data from/to Input/Output, memory and registers.
- Directs ALU, specifically to indicate the operations to be performed.

- The control unit operates according to the stored program, receiving and executing its instructions one at a time.

The Fetch-Execute Cycle (8)

Instruction Cycle (or Fetch-Execute Cycle)

- Fetch Cycle;
  - Fetch instruction from memory
- Execute Cycle;
  - Decode instruction,
  - Fetch required operands,
  - Perform operation.

The Fetch-Execute Cycle (9)

The components of a processor

- DR - Data Register
- M - Memory
- MAR – Memory Address Register
- MBR (or MDR) – Memory Buffer Register (or Mem Data Reg)
- AC - Accumulator
- PC - Program Counter
- IR - Instruction Register
- ALU - Arithmetic Logic Unit (including arithmetic logic circuits)
- PCU - Program Control Unit

The General Sequence to a Fetch-Execute Cycle

A program, stored in the main memory of a computer, is ‘obeyed’ by the CPU carrying out the following sequence of events, described with the symbolic notation:

1. [PC] -> MAR
2. [M] -> MBR
3. [PC]+1 -> PC
4. [MBR] -> IR
5. Decode
6. Execute this instruction
7. Repeat from step (1)

More on the Fetch-Execute Cycle

- Often the CPU actually has several tasks to perform.
- All modern computers use ‘multitasking’ to perform several tasks at once, devoting a fraction of a second to each task in turn.
- This application of multitasking is called ‘timesharing’. For example, the user might be typing a document while a clock is continuously displaying the time and a file is being downloaded from the internet.
- Each individual tasks of a CPU is called a ‘thread’.

To Include the ‘Store’ Operation

- Some books on Architecture prefer the ‘Fetch-Execute Cycle’ while others talk about a ‘Fetch-Execute-Store Cycle’.

- Here are a few slides to show the ‘Store’ feature… plus one more showing ‘ADD’ because the ADD operation is most common in the CPU.

Memory Operations

Two basic operations occur within this subcomponent: a fetch operation, and a store.

The fetch operation:

- A cell address is loaded into the Memory Address Register (MAR).
- The address is decoded, which means that through circuitry, a specific cell is located.
- The data contents contained within that cell are copied into another special register, called a Machine (or Memory) Data Register (MDR).
- Note that this operation is non-destructive – that is, the data contents are copied, but not destroyed.

Memory Operations (2)

The second memory operation is called a store.

- The fetch is like a read operation; the store is like a write operation.
- In the store, the address of the cell into which data are going to be stored is moved to the MAR and decoded.
- Contents from yet another special register, called an accumulator, are copied into the cell location (held in the MAR).
- This operation is destructive, meaning that whatever data were originally contained at that memory location is overwritten by the value copied from the accumulator.

Load Fetch-Execute Cycle

1. PC -> MAR	Transfer the address from the PC to the MAR
2. MDR -> IR	Transfer the instruction to the IR
3. IR(address) -> MAR	Address portion of the instruction loaded in MAR
4. MDR -> AC	Actual data copied into the accumulator
5. PC + 1 -> PC	Program Counter incremented

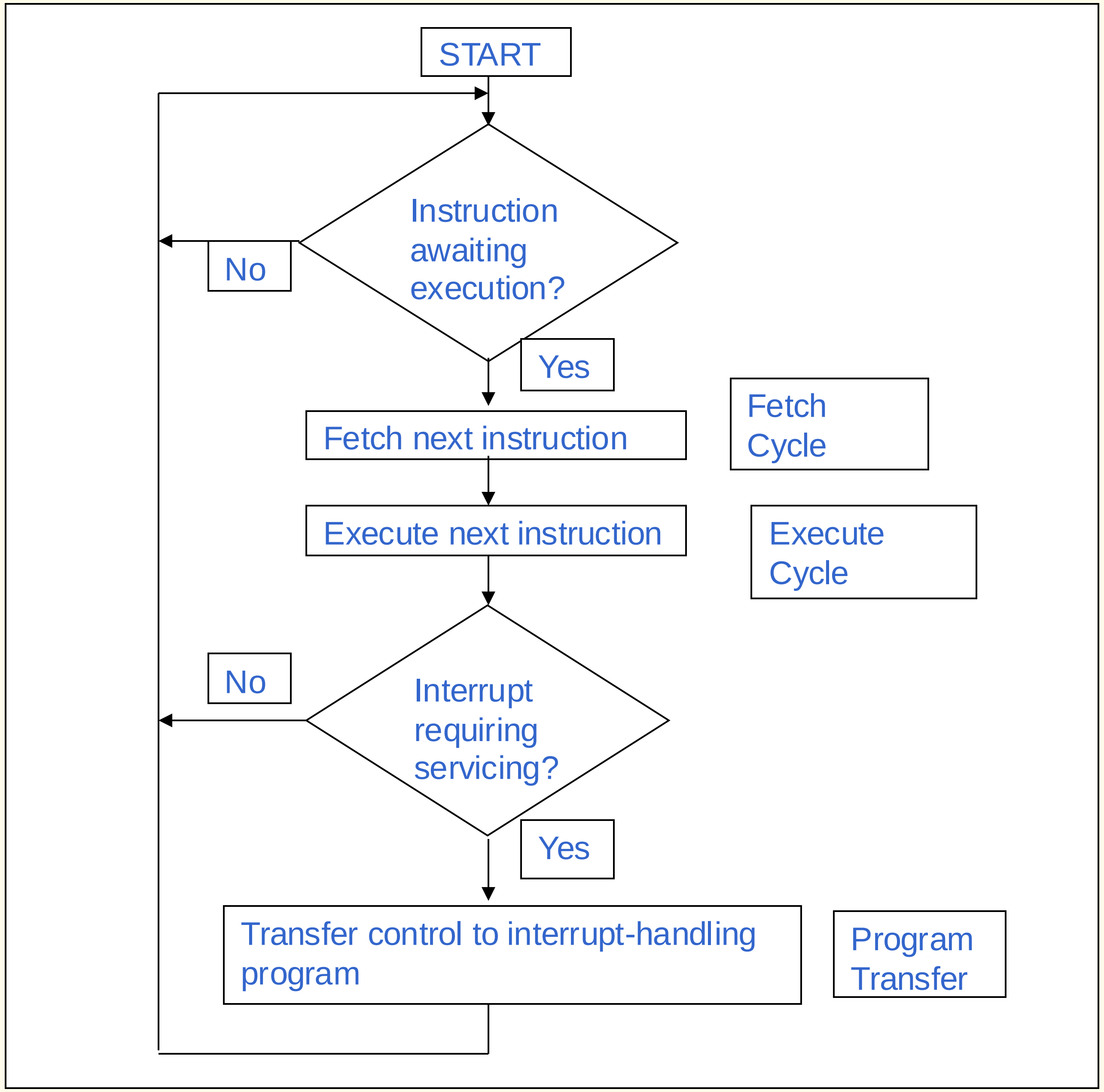
Store Fetch-Execute Cycle

1. PC -> MAR	Transfer the address from the PC to the MAR
2. MDR -> IR	Transfer the instruction to the IR
3. IR(address) -> MAR	Address portion of the instruction loaded in MAR
4. AC -> MDR	Accumulator copies data into MDR
5. PC + 1 -> PC	Program Counter incremented

Notice how Step 4 differs for LOAD and STORE

ADD Fetch-Execute Cycle

1. PC -> MAR	Transfer the address from the PC to the MAR
2. MDR -> IR	Transfer the instruction to the IR
3. IR(address) -> MAR	Address portion of the instruction loaded in MAR
4. AC + MDR -> AC	Contents of MDR added to the contents of the accumulator
5. PC + 1 -> PC	Program Counter incremented


START
Instruction awaiting execution?
No
Yes
Fetch next instruction
Fetch Cycle
Execute next instruction
Execute Cycle
Interrupt requiring servicing?
No
Yes
Transfer control to interrupt-handling program
Program Transfer

Overview of the Fetch-Execute Cycle

- RAM holds machine language programs and data.
- The CPU fetches machine language instructions from Cache or RAM (but possibly a 32 bit register) one after another and executes them without understanding what it does or how it does it.
- So the program it executes must be perfect, complete and unambiguous because the CPU can do nothing but execute instructions exactly as they are written.

Instruction and Data Movement

- The Fetch-Execute Cycle means that the processor must interact with registers, cache memory, RAM – and then a long list of other hardware input-output and storage devices.
- How do these strings of binary numbers move from device to device?
- They take the bus! (Buses, actually…)

Conclusion

The architecture of the modern computer has changed very little (at a fundamental level, at least) in the seventy or so years since it was first established.

The improvements and innovations that have extended from the classic architecture have been in memory structures and capacities, and in the way that the arithmetic/logic and control units function.

The innovation of computer architecture continues as time goes on.

End of von Neumann Architecture

- That describes a broad view of computer architecture in terms of the fundamental design known as von Neumann Architecture.
- We went into detail on some of the most fundamental features of the architecture and have seen how some of the subcomponents of the hardware system operate to support the overall computer system.
- The working principles of the central processing unit and internal memory have shown how instructions and data can be processed.

Are there… ANY QUESTIONS?

Where to Next?

WEEK 10:

The theme of the next lecture:

“Networks”

What is so great about networking computing devices? Are networks simple or complex? What sort of hardware and software is needed for a network? We can look at these things next.

Thanks for your attentiveness.

See you here next time. Be safe and well in the meantime.