COM1006 Devices and Networks (Autumn)
COM1090 Computer Architectures

Lecture #11

▶ The structure of the CPU

Dr Dirk Sudholt

Department of Computer Science

University of Sheffield

d.sudholt@sheffield.ac.uk

http://staffwww.dcs.shef.ac.uk/~dirk/campus_only/com1006/

Based on Chapter 7 in Clements, Principles of Computer Hardware

▶Aims of this lecture

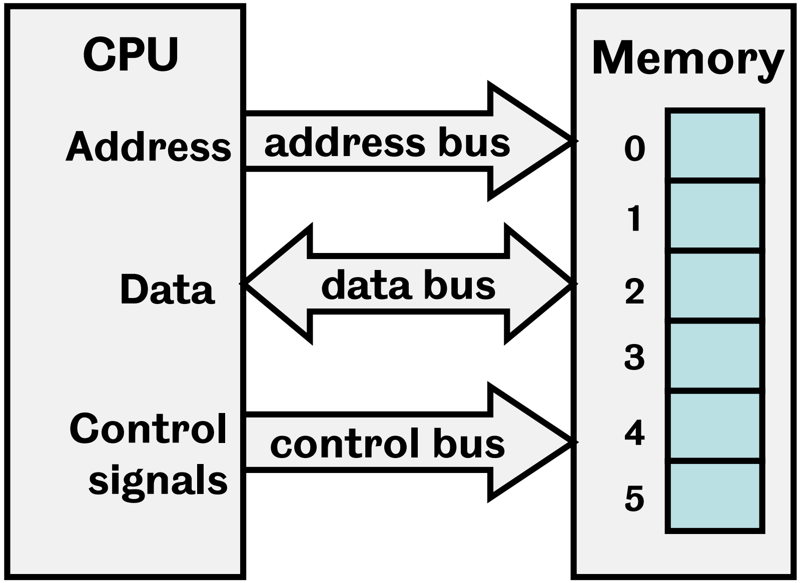
- To bridge the gap between digital logic and the instruction set architecture.
- To illustrate information flow on address paths and data paths in a simple CPU.
- To show how a computer converts an instruction opcode into actions that implement the opcode.
- To show how instructions are executed by control unit within the CPU.

▶Instruction execution

- This lecture bridges the gap between the instruction set architecture and digital circuits: we show how a computer instruction is **executed**.
- Assume a **one-and-a-half address architecture** in which one operand is in memory and the other is a register (the Motorola 68K is one such example).
- Consider the CPU in stages, adding more detail as we proceed.
- Highly simplified: show how instructions are executed from start to finish, but modern computers **overlap** the execution of instructions (pipelining – see later).

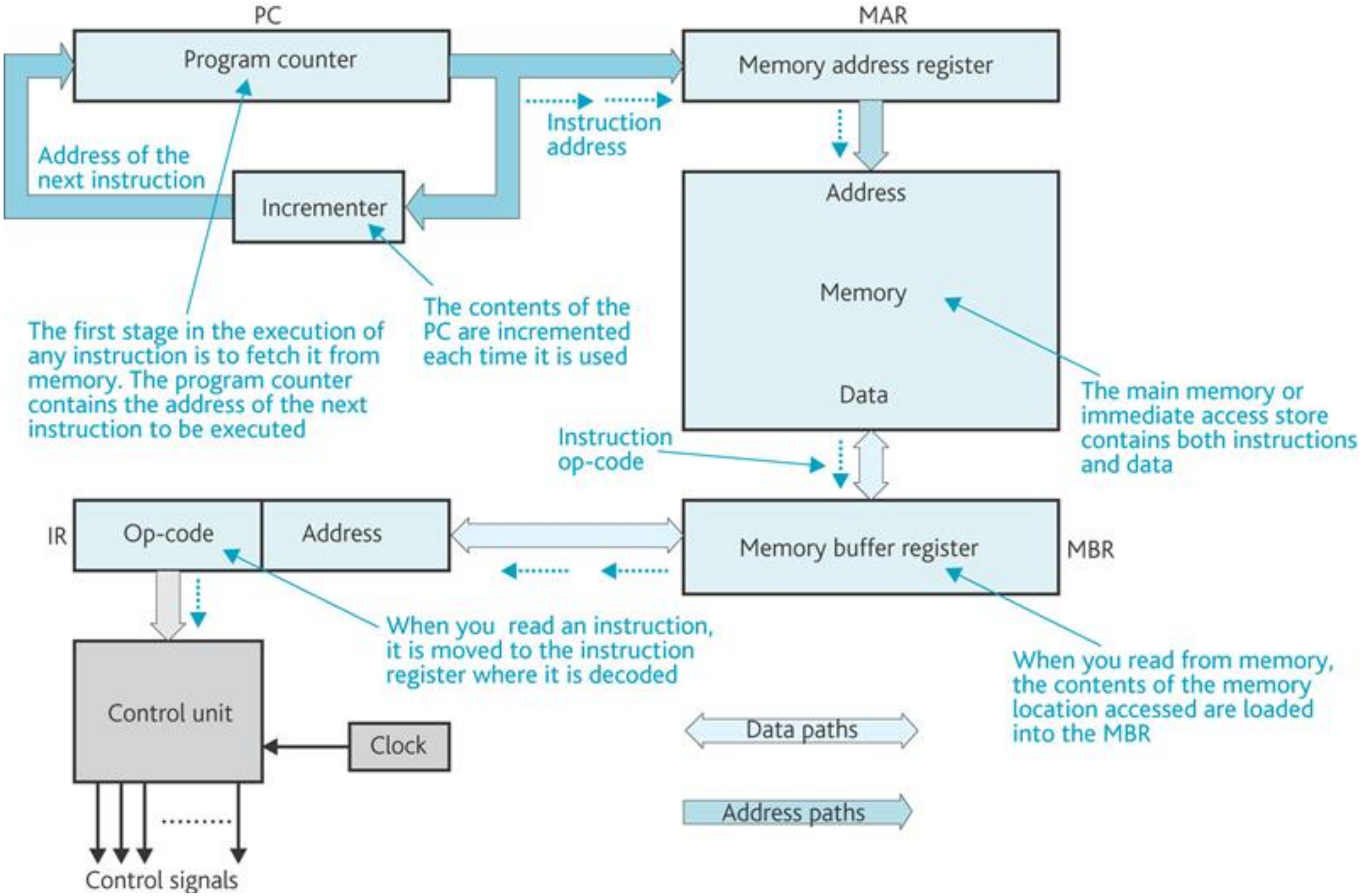
▶Information flow in the computer

- Three kinds of information flow:
  - **address paths** by which addresses flow from one part of the CPU to another
  - **data paths** by which instructions, constants, variables stored in memory and register flow
  - **control paths** which trigger events, provide clocks, control the flow of data and addresses



- First we consider address paths.
- Initially focus on the **fetch** phase of the fetch-execute cycle in which an instruction is read from memory.

CPU address paths


PC
Program counter
MAR
Memory address register
Instruction address
Address of the next instruction
Incrementer
Address
Memory
Data
The first stage in the execution of any instruction is to fetch it from memory. The program counter contains the address of the next instruction to be executed
The contents of the PC are incremented each time it is used
The main memory or immediate access store contains both instructions and data
Instruction op-code
IR
Op-code
Address
Memory buffer register
MBR
When you read an instruction, it is moved to the instruction register where it is decoded
When you read from memory, the contents of the memory location accessed are loaded into the MBR
Control unit
Clock
Data paths
Address paths
Control signals

▶Reading the instruction: registers etc.

- **Program counter** (PC) points to the next instruction to be executed.
- **Memory address register** (MAR) hold address of memory location into which data is being written in a write cycle, or from which data is being read during a read cycle.
- **Memory buffer register** (MBR) temporary holding place for data that is read from, or written to, memory.
- **Instruction register** (IR) contains instruction which is split into two fields, the operation code (**opcode**) and **operand** field (address or constant).
- **Control unit** (CU) takes opcode and stream of clock pulses and generates signals to control the CPU.

▶Fetch phase summarised in RTL

$[MAR] \leftarrow [PC]$ Copy contents of PC to MAR.

$[PC] \leftarrow [PC]+4$ Increment contents of PC so that it points to the next instruction to be executed.

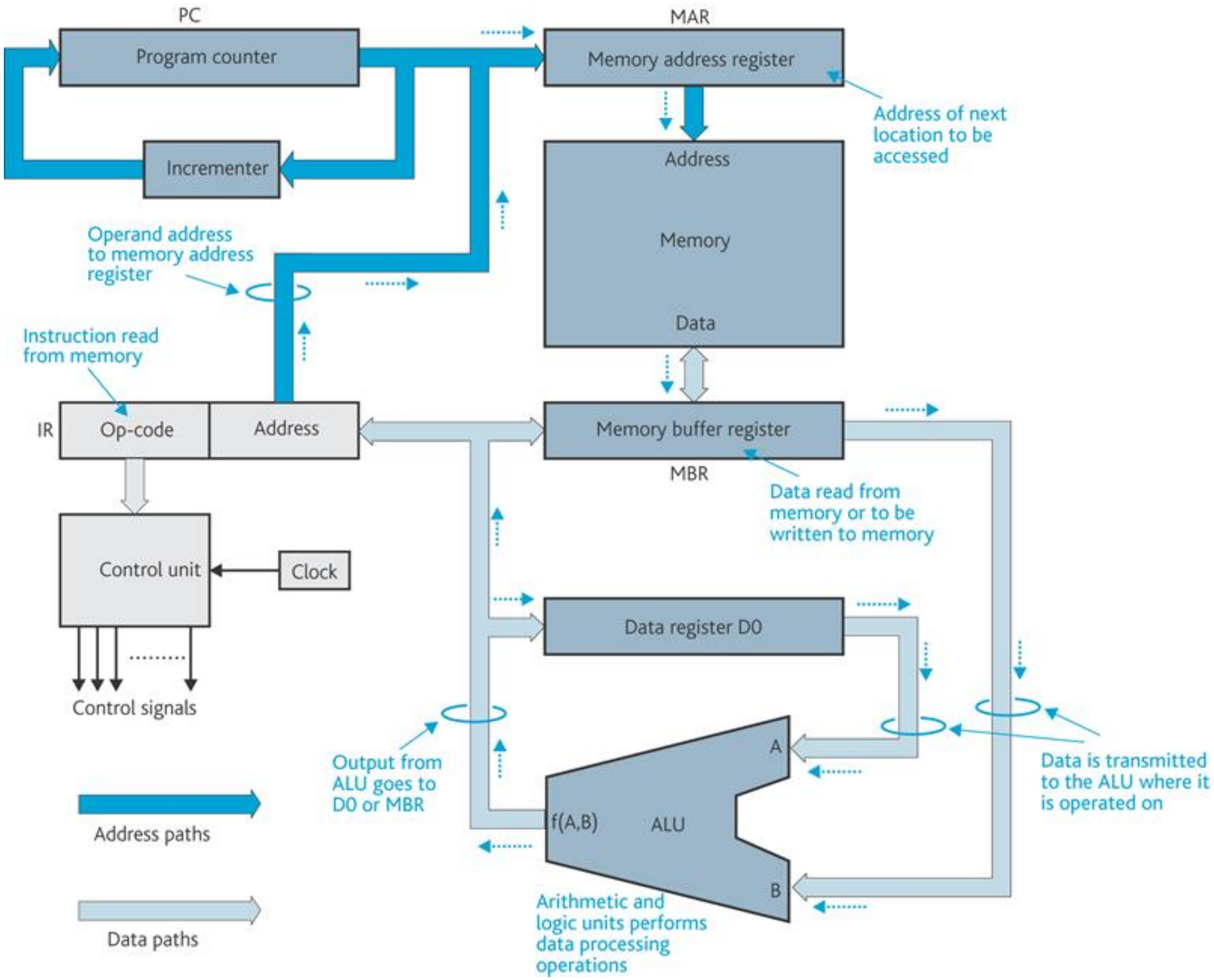
$[MBR] \leftarrow [[MAR]]$ MAR contains copy of (previous) PC. A **memory read cycle** is performed that reads the instruction from address pointed to by MAR into the MBR.

$[IR] \leftarrow [MBR]$ Instruction is moved from MBR to IR, where it is divided into opcode and address.

$CU \leftarrow [IR_{(opcode)}]$ Opcode is copied to CU which coordinates execution of instruction (e.g., by ALU)

Note: assumes that instructions are all 32 bits long. This is true of RISC but not CISC processors such as 68K and Pentium which have variable length instruction formats.

CPU data paths


PC
Program counter
MAR
Memory address register
Address of next location to be accessed
Incrementer
Address
Memory
Data
Operand address to memory address register
Instruction read from memory
IR
Op-code
Address
Memory buffer register
MBR
Data read from memory or to be written to memory
Control unit
Clock
Control signals
Data register D0
Output from ALU goes to D0 or MBR
A
f(A,B)
ALU
B
Data is transmitted to the ALU where it is operated on
Arithmetic and logic units performs data processing operations
Address paths
Data paths

Explaining the data paths

- **Data register** holds temporary results during a calculation. Called D0 for compatibility with 68K.
- Also called the **accumulator** (A) in one-address machines.
- CPU design can be easily extended to multiple registers.
- **Arithmetic and logical unit** (ALU) carries out calculations. Takes input and directs output to/from D0 and/or MBR.

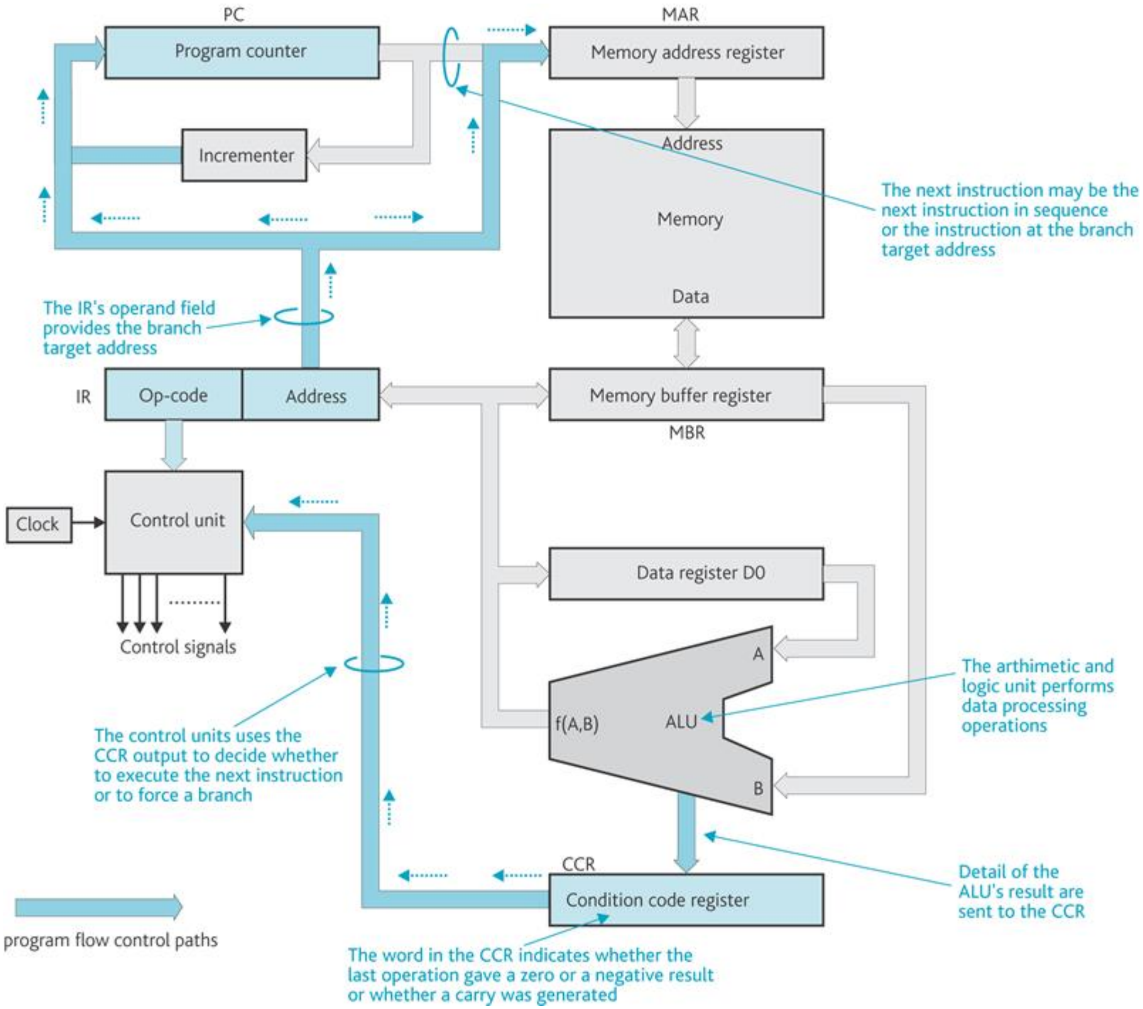
Execution of ADD P, D0 ([D0]←[D0]+[P])

FETCH	$[\text{MAR}] \leftarrow [\text{PC}]$	See slide 7
	$[\text{PC}] \leftarrow [\text{PC}]+4$	
	$[\text{MBR}] \leftarrow [[\text{MAR}]]$	
	$[\text{IR}] \leftarrow [\text{MBR}]$	
	$\text{CU} \leftarrow [\text{IR}_{(\text{opcode})}]$	
ADD	$[\text{MAR}] \leftarrow [\text{IR}_{(\text{address})}]$	Move operand address to MAR
	$[\text{MBR}] \leftarrow [[\text{MAR}]]$	Read data from memory
	$\text{ALU} \leftarrow [\text{MBR}],\ \text{ALU} \leftarrow [\text{D0}]$	Perform the addition (two operations are executed simultaneously)
	$[\text{D0}] \leftarrow \text{ALU}$	Move result to data register

Executing conditional instructions

- So far we've considered sequential flow of control; also need to implement **conditional branches** and **jumps** such as `BNE Loop`.
- To do this we need:
  - Condition code register (CCR) or processor status register. Records the carry, negative, zero and overflow flag bits.
  - A path between the CCR and the control unit so that instructions can interrogate the CCR.
  - A path between the address field of the instruction register and the program counter so that the PC can be changed as the result of a branch or jump instruction.

Info paths for conditional instructions



The next instruction may be the next instruction in sequence or the instruction at the branch target address

The IR's operand field provides the branch target address

The arthimetic and logic unit performs data processing operations

The control units uses the CCR output to decide whether to execute the next instruction or to force a branch

Detail of the ALU's result are sent to the CCR

The word in the CCR indicates whether the last operation gave a zero or a negative result or whether a carry was generated

Example: conditional instruction

- A conditional branch instruction tests a status bit of the CCR. Then either
  - The next instruction is fetched in the usual way, **or**
  - The next instruction is obtained from the location whose **target address** is in the instruction register.
- Example: Branch on equal to zero (jump to target address if the Z-bit in the CCR is 1)
- In RTL:

`BEQ target:` IF [Z]=1 THEN [PC] ← $[IR_{(address)}]$

▶Dealing with literal operands

- Sometimes a literal operand value is supplied (this is indicated as part of the opcode).
- This requires a data path between the operand field of the IR and the data register for instructions such as

```
MOVE #30, D0
```

$[D0] \leftarrow 30$

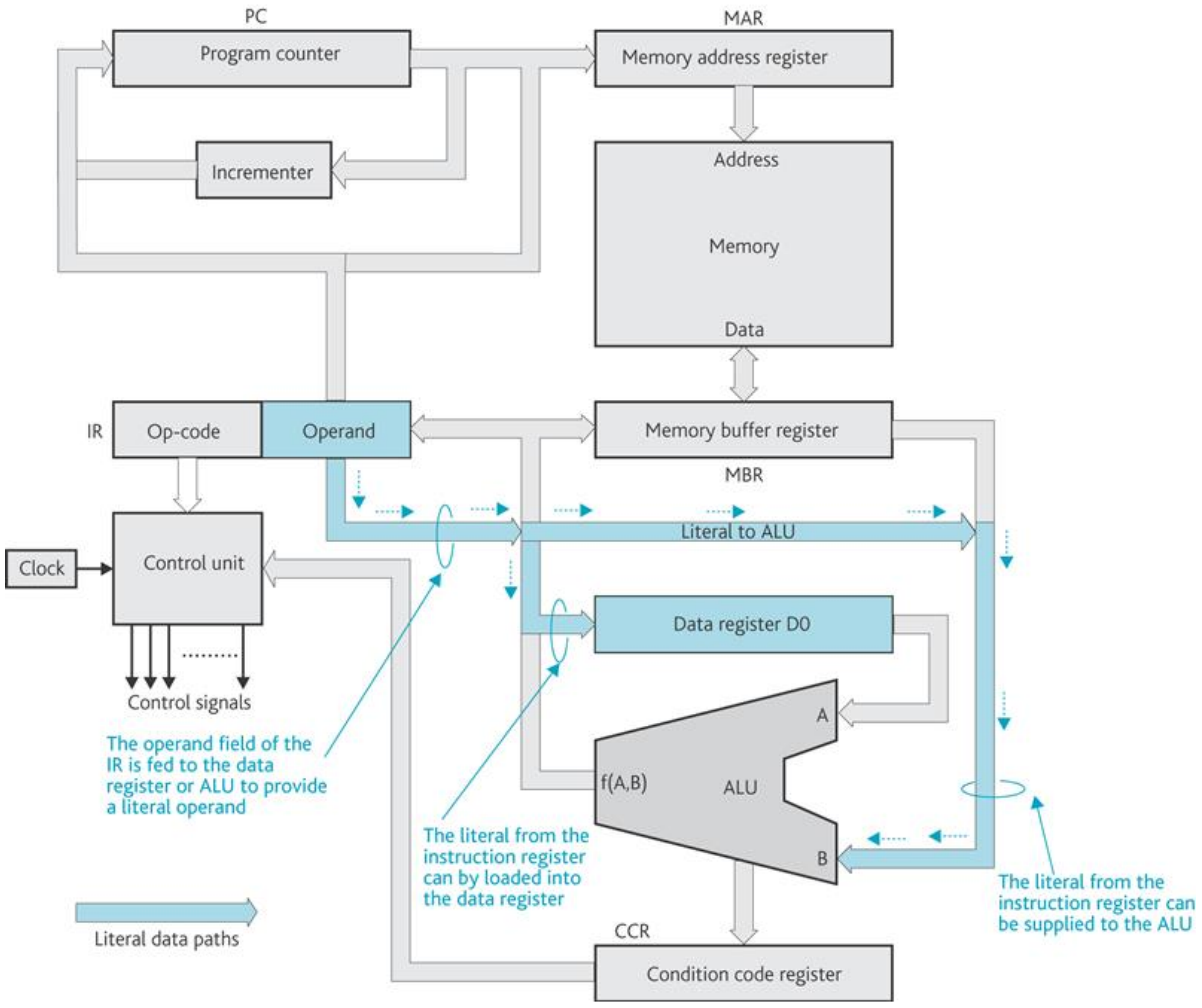
and a data path between the operand field of the IR and the ALU for instructions such as

```
ADD #12, D0
```

$[D0] \leftarrow [D0] + 12$

- The schematic CPU shown next is essentially complete; it can execute any computer program (but enhancements can improve performance, such as adding more registers).

Complete schematic of simple CPU



The operand field of the IR is fed to the data register or ALU to provide a literal operand

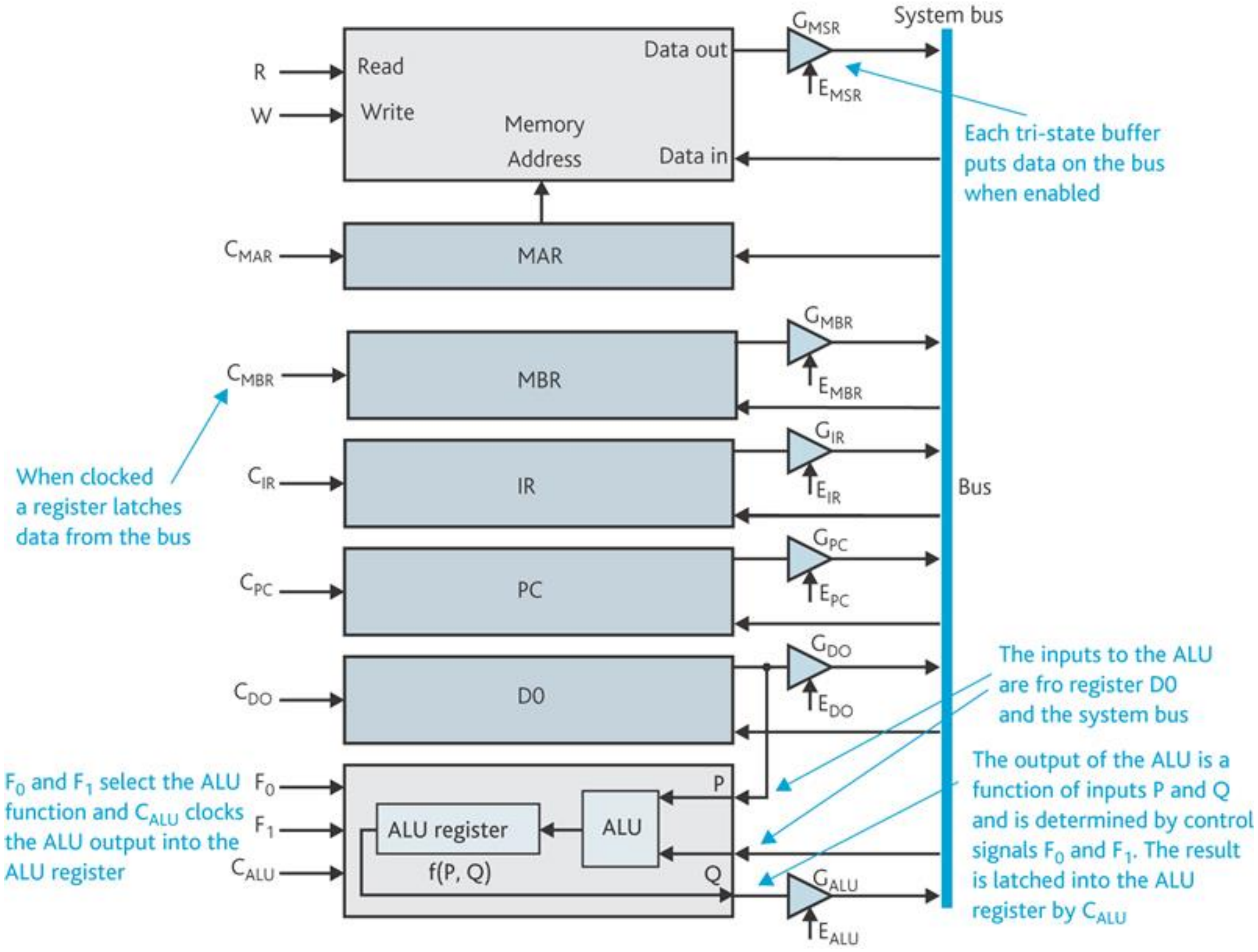
The literal from the instruction register can by loaded into the data register

The literal from the instruction register can be supplied to the ALU

How is an instruction executed?

- How is a binary pattern (e.g., 1101101110101001_2) turned into a sequence of machine-level instructions that cause an instruction (e.g., ADD P, D0) to be executed?
- Control unit provides logic that directly transforms instructions into control signals (**random logic control unit**)
- **Random** logic control unit implies that arrangement of gates is **specific** for a particular computer.
- Opcode is decoded by a **demultiplexer**.
- To illustrate a random control unit, we consider a primitive CPU and the control signals needed to operate it.

Simple single-bus CPU



Each tri-state buffer puts data on the bus when enabled

When clocked a register latches data from the bus

The inputs to the ALU are fro register D0 and the system bus

The output of the ALU is a function of inputs P and Q and is determined by control signals F_0 and F_1. The result is latched into the ALU register by C_{ALU}

F_0 and F_1 select the ALU function and C_{ALU} clocks the ALU output into the ALU register

▶Information flow in the simple CPU

- Simple CPU with bus connecting registers and memory.
- Memory receives address of location to be accessed from MAR. Note that data can be transferred directly to register from memory (not only via MBR).
- ALU has two inputs (P from D0, Q from bus).
- ALU controlled by a two-bit code (F_0 and F_1), e.g. 00 → add P to Q, 01 → subtract Q etc. ALU has an internal register.
- When tri-state gates (e.g., G_{MSR}) are enabled, data flows from memory or register onto bus.
- Data is copied from bus to register by clocking it (e.g., C_{IR})

Control signals in the simple CPU

Signal	Type	Operation
R, W	Memory control	Read from / write to memory
C_{MAR}	Register clock	Clock data into MAR
C_{MBR}	Register clock	Clock data into MBR
C_{PC}	Register clock	Clock data into program counter
C_{IR}	Register clock	Clock data into instruction register
C_{D0}	Register clock	Clock data into data register
C_{ALU}	Register clock	Clock data into ALU register
E_{MSR}	Bus control	Gate data from memory onto the bus
E_{MBR}	Bus control	Gate data from MBR onto the bus
E_{IR}	Bus control	Gate operand from IR onto the bus
E_{PC}	Bus control	Gate program counter onto the bus
E_{D0}	Bus control	Gate data register onto the bus
E_{ALU}	Bus control	Gate ALU function register onto the bus
F_0, F_1	ALU control	Select ALU function

Example: Implementing `LOAD N`

- Consider a LOAD N instruction [D0] ← [N].
- First we define the operations (in RTL) needed to perform this instruction

[MAR] ← [IR]	copy target address (N) from IR to MAR
[D0] ← [[MAR]]	copy data at target address to data register

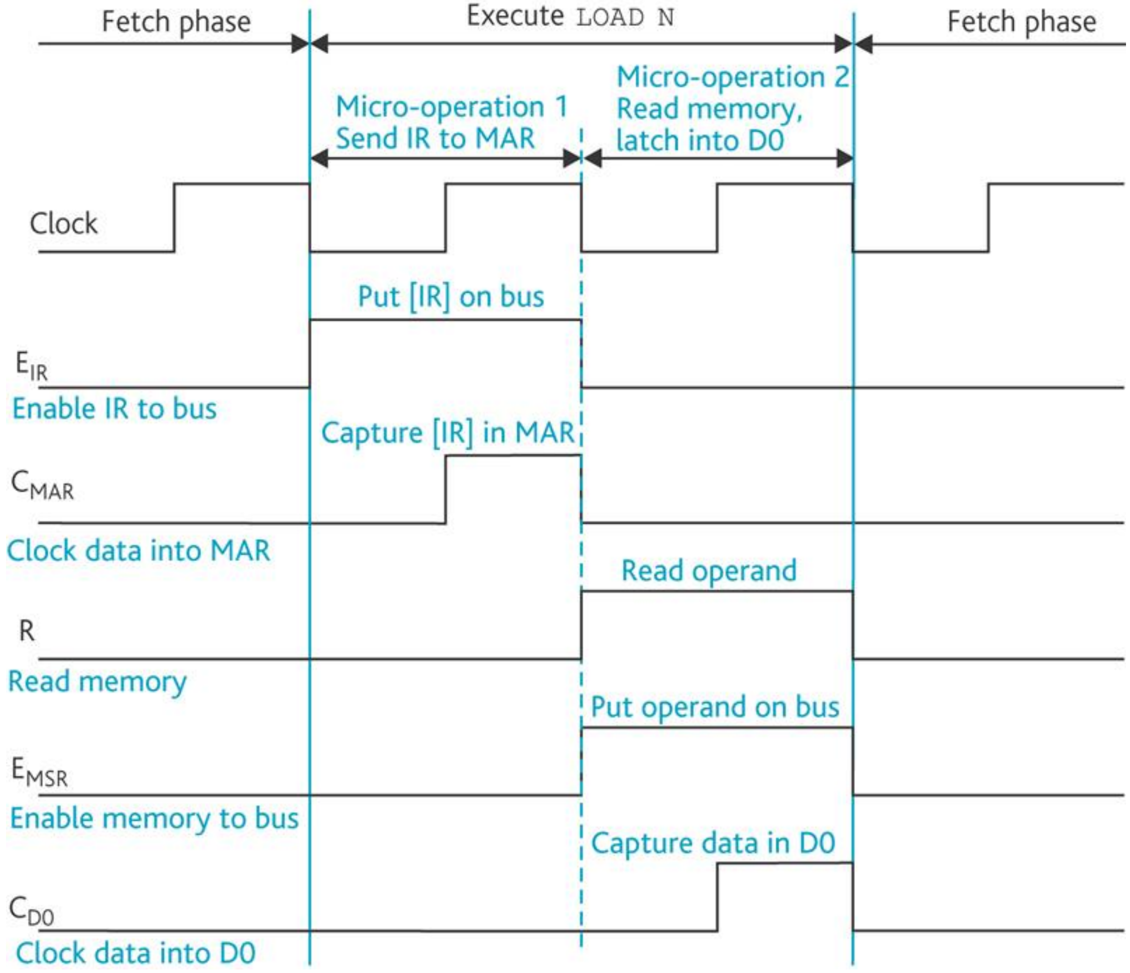
- Now consider the control signals needed to implement each step in the simple CPU:

[MAR] ← [IR]	$E_{IR}=1$, $C_{MAR}=1$
[D0] ← [[MAR]]	$R=1$, $E_{MSR}=1$, $C_{D0}=1$

- Combine with **timing signals** to do things at the right time.

▶Timing of the execute phase of `LOAD N`

Note: other control signals are not shown because they do not play a role in the execution of the `LOAD N` instruction



Summary

- Learned how information flows in a CPU.
- The CPU consists of
  - data paths and address paths
  - registers and an interface to memory (MAR, MBR)
  - an arithmetic logic unit (ALU)
  - a control unit which controls clock signals and data paths.
- An instruction (c.f. assembly language mnemonic) is implemented by a sequence of microinstructions.
- Instructions have a fetch cycle and an execute cycle.
- A random logic control unit implements macroinstructions directly in digital logic.