

If you didn't take COMP15111 (ARM assembly code) you should be in IT407

Before we start:

What is a CPU?

What is RAM?

What is a register?

What is an assembly-language instruction?

If you didn't take COMP15111 (ARM assembly code) you should be in IT407

COMP25111: Operating Systems

Lecture 2: Computer Architecture – MU0 Datapaths

John Gurd

School of Computer Science, University of Manchester

Autumn 2015

Overview & Learning Outcomes

Datapath & Control

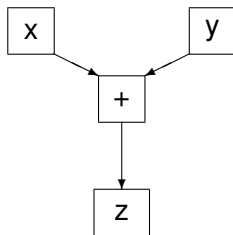
ISA of a very simple computer: MU0

Building MU0 (Lab exercise 1)

Datapath

CPU acts on fixed sized items of data (e.g. words)
moved around bit-parallel (i.e. a connection for each bit).

Operations on each data bit tend to be identical,
so the hardware logic is duplicated
and the datapath is very regular



Control

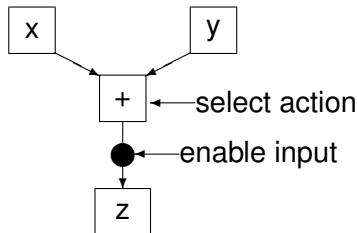
Logic which determines the flow of data through the datapath.

Governs which operation is performed and when.

Different for each instruction in the instruction set.

So not regular and more difficult to design.

Start with the datapath, then derive the control logic.



MU0 ISA

“One-address” instructions

Word size = 16 bits

4K (4096, 2^{12}) words of memory

Registers: Program Counter, Accumulator
(small, fast memory within the CPU)

PC (12 bits) – Program Counter
Address of next instruction to execute

ACC (16 bits) – Accumulator
Result of last load or arithmetic operation

Instruction Set

Encoding	Format	Meaning
0000	LDA s	ACC = [s]
0001	STA s	[s] = ACC
0010	ADD s	ACC += [s]
0011	SUB s	ACC -= [s]
0100	JMP s	PC = s
0101	JGE s	if ACC \geq 0 then PC = s
0110	JNE s	if ACC \neq 0 then PC = s
0111	STP	halt execution
1???	not used	

Q: what does this do?

```
loop: LDA x
      ADD y
      STA x
      LDA z
      SUB one
      STA z
      JNE loop
      STP

x:    0
y:    4
z:    3
one:  1
```


Q: without using a loop, multiply by 10

Processor State

The contents of the registers.

If we wanted to interrupt the processor (e.g. to do something else) saving the registers and later reloading them would allow the program to continue as normal.

This is true no matter how many registers (e.g. ARM)

Q: Caveat?

Processor phases

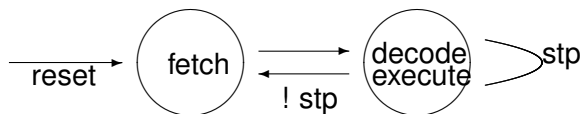
Assume simple (un-cached) connection: RAM \Leftrightarrow CPU

All instructions fetched from memory before being executed

Some instructions access memory during execution (e.g. LDA, STA)

Decision: 1 processor phase per (potential) memory access

So 2 phases (each 1 clock cycle): **fetch** and **execute**



fetch

All instructions do the same thing during fetch:

- Use PC as address to read memory
- Save result of read in CPU (where?)
- Increment PC (use a special purpose adder)

so need: IR (16 bits) – Instruction Register
(The current instruction being executed)

so need datapaths: $\text{RAM} \Leftarrow \text{PC}$, $\text{IR} \Leftarrow \text{RAM}$

(and control signal to read from RAM)

execute

Different for each instruction.

So control depends on the opcode (function) bits of IR (IR.F)

So now we examine each instruction (type)

JMP operation

$PC = s$

get s from bottom 12 bits of IR (IR.S)
copy it over current contents of PC

so need datapath: $PC \leftarrow IR.S$

STA operation

[s]= ACC

get s from IR.S and send it to RAM (address)

get contents of ACC and send it to RAM (data)

perform the write in RAM

so need datapaths:

RAM (address) \Leftarrow IR.S

RAM (data) \Leftarrow ACC

(and control signal to write to RAM)

ADD operation

ACC += [s]

get s from IR.S and send it to RAM

get result from RAM and send it to an adder

get contents of ACC and send it to other adder input

perform the addition and send the result to the ACC

so need Arithmetic & Logic Unit (ALU) within CPU

so need datapaths:

ALU \leftarrow ACC

ALU \leftarrow RAM

ACC \leftarrow ALU

(and control signal to tell ALU to add)

Q

SUB operation?

LDA operation?

Timing

e.g. how do we make “ADD” work properly,
as ACC is both output to and input from the ALU?

- start of clock cycle: allows data to propagate out from registers
- send control signals for RAM or ALU operation etc.
- allow time for operation to happen
- end of clock cycle: copies (enabled) inputs into registers

(lots of electronic details to make sure this works)

Designing Datapaths

Put the paths above together:

- PC or RAM (address) \Leftarrow IR.S
- RAM (address) \Leftarrow PC
- RAM (data) or ALU (left) \Leftarrow ACC
- ALU (right) or IR \Leftarrow RAM
- ACC \Leftarrow ALU

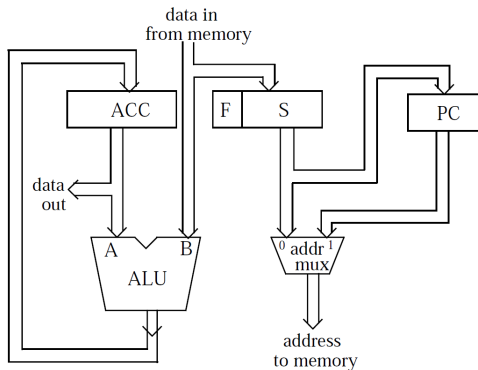
Values from one source can go to multiple destinations

Values to one destination can only come from one source

Use multiplexers to resolve multiple sources:

- RAM (address) \Leftarrow Multiplexer \Leftarrow PC or IR.S

MU0 Datapaths



Summary of key points

Datapath – values flow around CPU

Control – logic to control flows & perform actions

ISA of a very simple computer: MU0

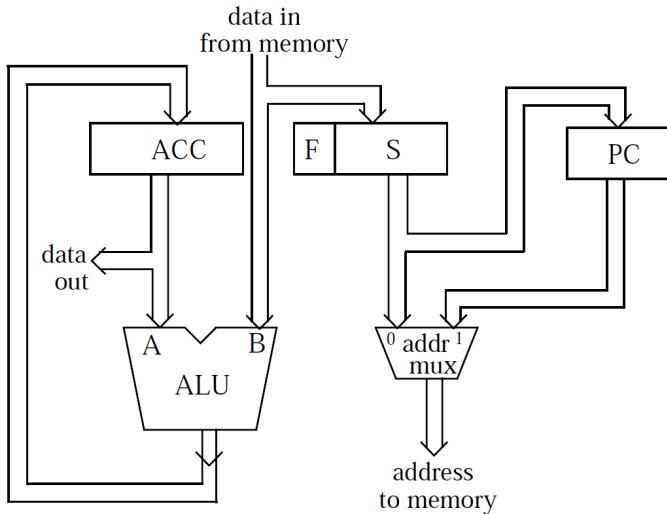
Building MU0: Datapaths

- simple link CPU \Leftrightarrow RAM
- 2 clock cycles per instruction: Fetch, Execute
- extra H/W: IR, ALU, Multiplexer, PC+=1
- cheap: reuse datapaths

Your Questions

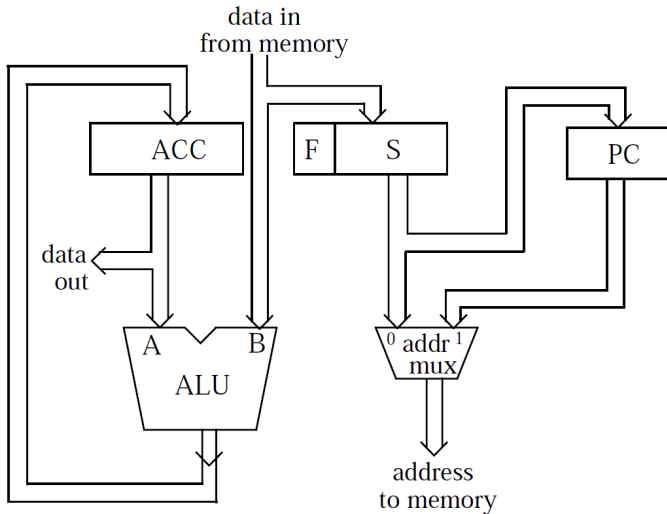
For next time – LDA execute phase

Shade in the path usage for the execute phase for the LDA instruction on the MU0 datapath diagram below:



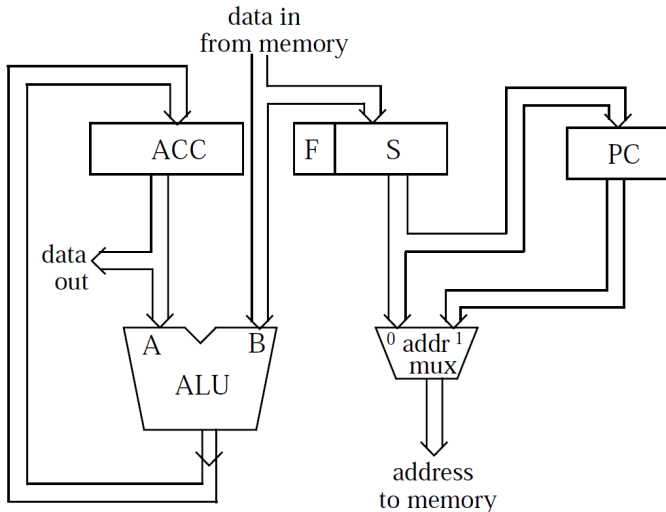
For next time – STA execute phase

Shade in the path usage for the execute phase for the STA instruction on the MU0 datapath diagram below:



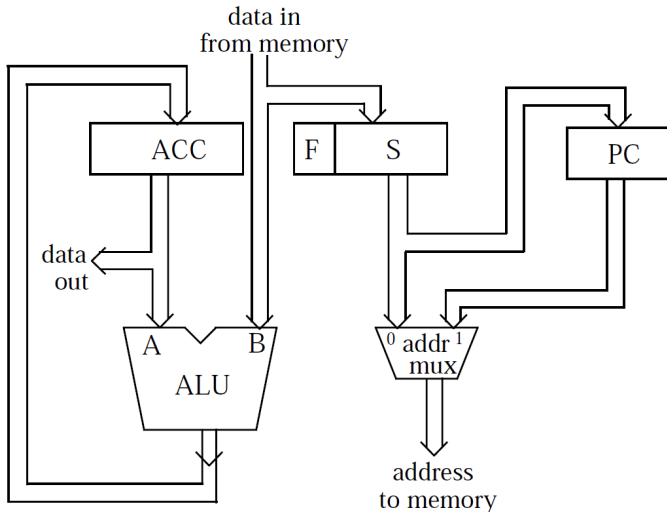
For next time – JMP execute phase

Shade in the path usage for the execute phase for the JMP instruction on the MU0 datapath diagram below:



For next time – ADD execute phase

Shade in the path usage for the execute phase for the ADD instructions on the MU0 datapath diagram below:



Glossary

Instruction Set Architecture (ISA)

Processor State

Fetch

Execute

Processor Phase

Logic

Signal

Control

Datapath

ALU

Multiplexer

Reading

http://www.cs.man.ac.uk/~pjj/cs1001/mu0_lab.html

<http://www.cs.man.ac.uk/~pjj/cs1001/arch/index.html>

COMP12111 lecture handouts for “Processors” via

<http://www.cs.manchester.ac.uk/ugt/COMP12111/>

Looking at Intel’s Prescott die:

<http://www.chip-architect.com/news/>

[2003_03_06_Looking_at_Intels_Prescott.html](http://www.chip-architect.com/news/2003_03_06_Looking_at_Intels_Prescott.html)

[2003_04_20_Looking_at_Intels_Prescott_part2.html](http://www.chip-architect.com/news/2003_04_20_Looking_at_Intels_Prescott_part2.html)

[Northwood_130nm_die_text_1600x1200.jpg](http://www.chip-architect.com/news/Northwood_130nm_die_text_1600x1200.jpg)