MANCHESTER 1824
The University of Manchester

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

# COMP25111: Operating Systems

## Lecture 3: Computer Architecture – MU0 Control Signals

Will Toms

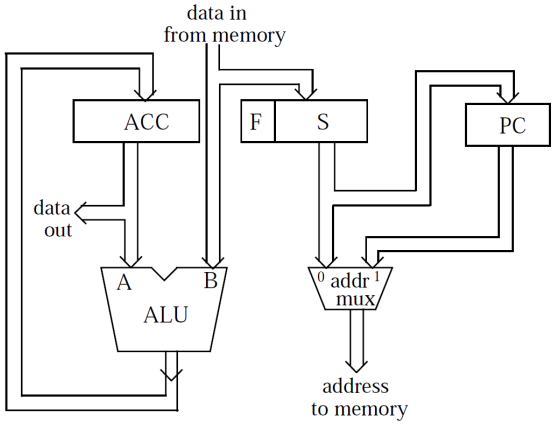
School of Computer Science, University of Manchester

Autumn 2016



## From last time – fetch phase

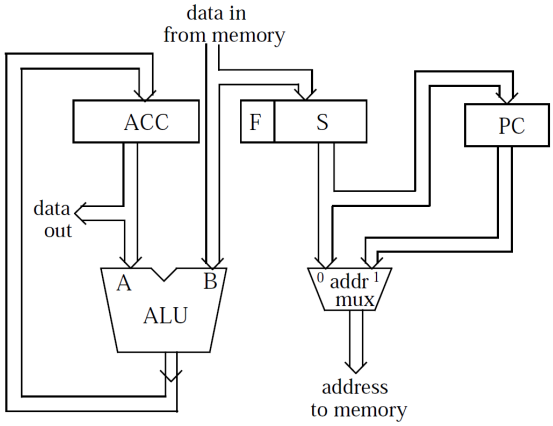
Shade in the path usage for the fetch phase on the MU0 datapath diagram below:






## From last time – LDA execute phase

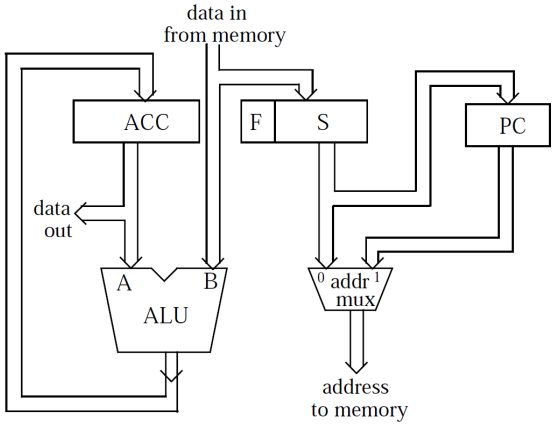
Shade in the path usage for the execute phase for the LDA instructions on the MU0 datapath diagram below:






## From last time – STA execute phase

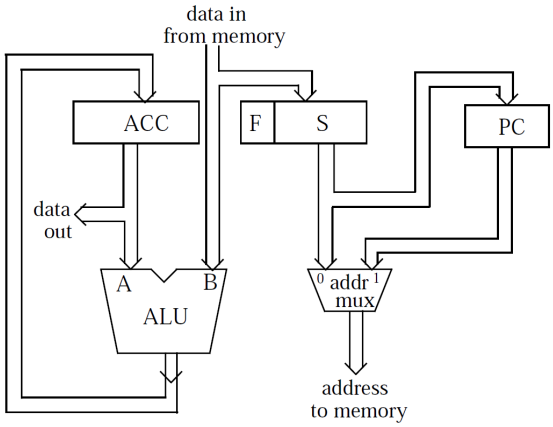
Shade in the path usage for the execute phase for the STA instruction on the MU0 datapath diagram below:






## From last time – JMP execute phase

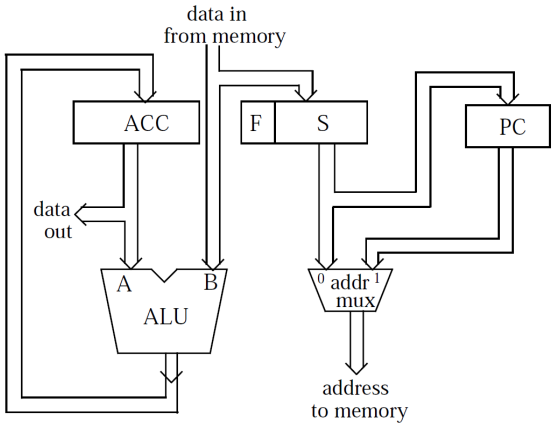
Shade in the path usage for the execute phase for the JMP instruction on the MU0 datapath diagram below:






## From last time – ADD execute phase

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



## Overview & Learning Outcomes

MU0 Control Signals

Introduction to lab 1

Verilog



## What Control Signals?

Push: values from registers & devices always available
Control the actions registers & devices perform on inputs

Each register needs an enable, to allow it to be written to: En_ACC, En_PC, En_IR

Multiplexer needs a signal to select an input

Memory needs actions: Ren (Read Enable) & Wen (Write Enable)

ALU needs actions: add, sub, & byp (bypass)



## ALU control signals

We are using three signals (add, sub & byp)

At most one of the three can be set during any phase.

In theory two (encoded) bits would do

Decode either in Control (3 signals) or in ALU (2 signals)



## MU0 Control

A “black box”

with these inputs:
external signals: clock reset
from datapaths: F[3:0] N Z
(N & Z come from ACC, needed for JGE & JNE)

and these outputs:
within CPU: En_IR En_PC En_ACC byp add sub addr_Mux
to memory: Ren Wen



## Lab 1: What is inside the “black box”?

Must create output signals for each phase of each instruction

These derive from the 6 input bits (F[3:0] N Z)

Some outputs can be “don’t care”
(may simplify the logic circuit)

e.g. ALU action during a JMP

(indicate by “X” instead of “1” or “0”)

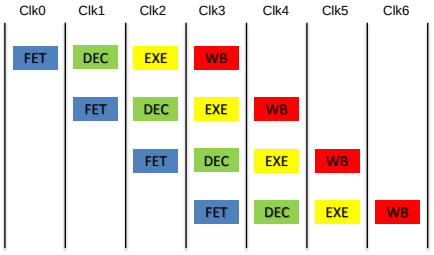
Never make register enables or Wen don’t care!



MANCHESTER 1824
The University of Manchester

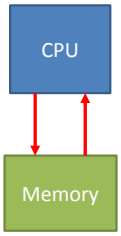
## Pipelining

- Speed up execution by operating phases in parallel
- Pentium 4 had 30 pipeline stages
- Power consumption too great
- Multiple simplified cores (Core2 and sucessors)



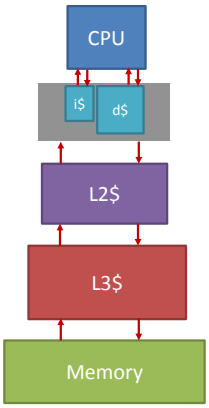
## Von Neumann Architecture

- Unified Memory Model
- Instructions and Data reside in the same space
- CPU reads and writes directly to memory



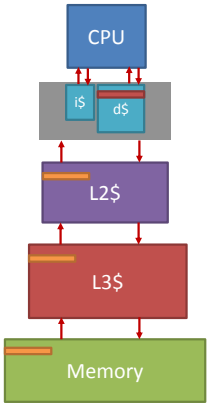

## Caches

- Memory much slower than CPU
- Caches exploit
  - Temporal Locality
    - Locations used more than once
  - Spatial Locality
    - Things closed together used at the same time



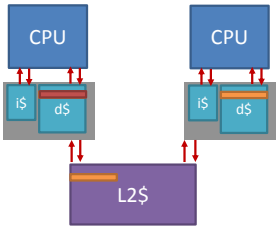

## Caches

- Line Size 64 Bytes
- Hit Latency:
- L1i 1 Cycle
- L1d 2 Cycles
- L2 12 Cycles
- L3 20 Cycles
- Main Memory ~100cycles
- Written (dirty) lines can reside in the cache until they need to be written back to memory




## Cache Coherence

- With multiple cores copies can be out of sync
- Must *invalidate* other copies before write
- Different applications can overwrite each others data
- Some applications caching not beneficial



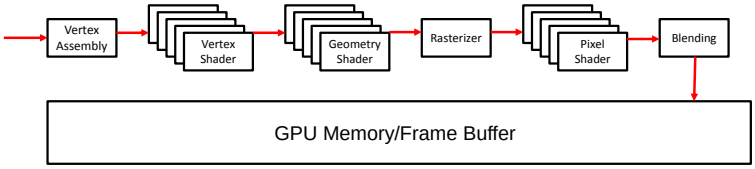

## Energy-Efficiency

- CPUs unlikely to go faster
- ICs unlikely to get bigger
- Power Consumption biggest factor
  - Data Centre's account for 2% of global energy usage
  - Battery Life biggest limiting factor in mobile computing
- Technology will become more efficient
- Specialised compute resources for common tasks (*accelerators*)

## Energy-Efficiency

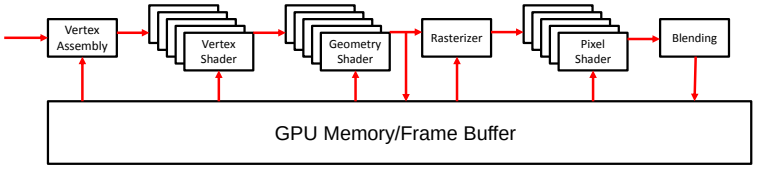
- CPUs unlikely to go faster
- ICs unlikely to get bigger
- Power Consumption biggest factor
  - Data Centre's account for 2% of global energy usage
  - Battery Life biggest limiting factor in mobile computing
- Technology will become more efficient
- Specialised compute resources for common tasks (accelerators)

## GPUs



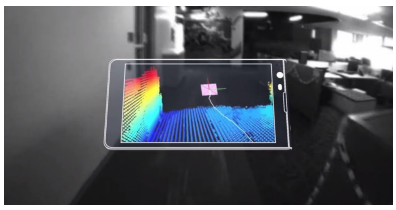

- Graphics Coprocessors have been around since 1980's
- Multi-stage piplines with many parallel multipliers
  - Great for matrix multiplication tasks

## GPGPUs




- Iterative Pipelines allow general purpose computations
- GPUs now coherent
- Difficult to program

## Accelerators

TensorFlow

Tango

## Post Von-Neumann?

- Von Neumann model still remain
- Iteraction with accelerators via API's/Data-Types and Design Patterns
- Not all designs will map well to heregeneous architectures
- **Data Movement** most expensive operation