MANCHESTER 1824

The University of Manchester

Asynchronous execution: one instruction's journey through a clock-free microprocessor.

Jim Garside

University of Manchester

http://www.cs.manchester.ac.uk/apt/projects/processors/amulet/AMULET3i.php

Advanced Processor Technologies Group

Contents

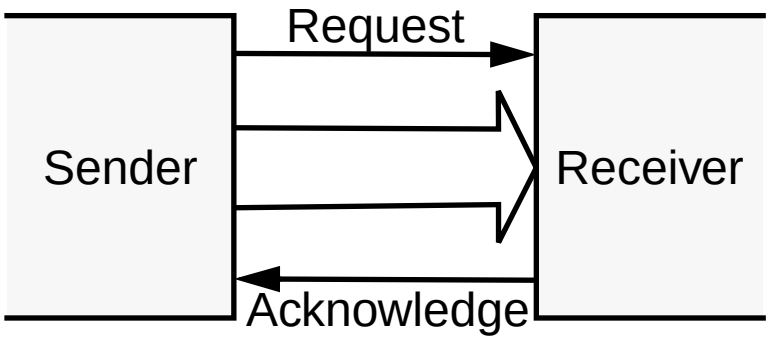
- Why asynchronous? How's it done?
- Why an instruction's view?
- Which instruction set? Which instruction?
- Processor overview
- Down the pipeline
  - Fetch
  - Thumb
  - Decode
  - Execute
  - Memory
  - Reorder/writeback
- A bit about the chip
- Conclusions

Why asynchronous?

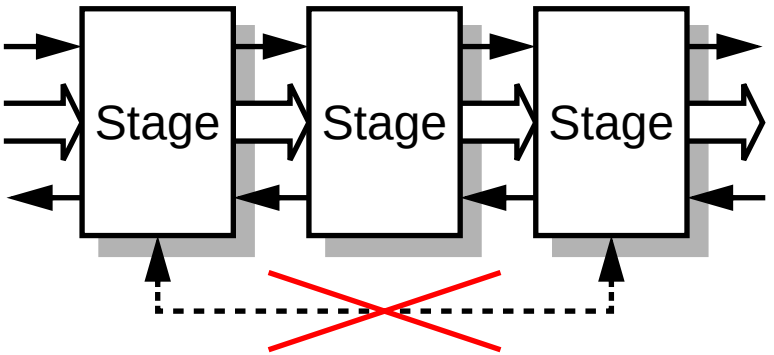
- Low power
  - equivalent of clock gating on every subcircuit
  - reduce power merely by reducing supply voltage
- Low EMI
  - no coherence provided by clock edge
- Composability
  - all interactions on local basis; nothing ‘global’
- Some other things
  - e.g. timing closure on a system basis
- Because it wasn’t there

How is it done?

By handshaking …



- Message passing
- Rendezvous only when necessary



- Form into pipelines
- Non-local action disallowed (mostly)

Why an instruction's view?

- Gives a different perspective to the system
- Justifies illustration of a variety of different mechanisms
- Hopefully coherent!

Which ISA?

- Amulet processors have all been based on ARM architecture
- The microarchitecture described here is Amulet3
  - equivalent to ARM9
  - full featured/fully instruction set compatible
- Relevant ARM features:
  - PC in register set
  - precise aborts
  - Thumb expansion

Which instruction?

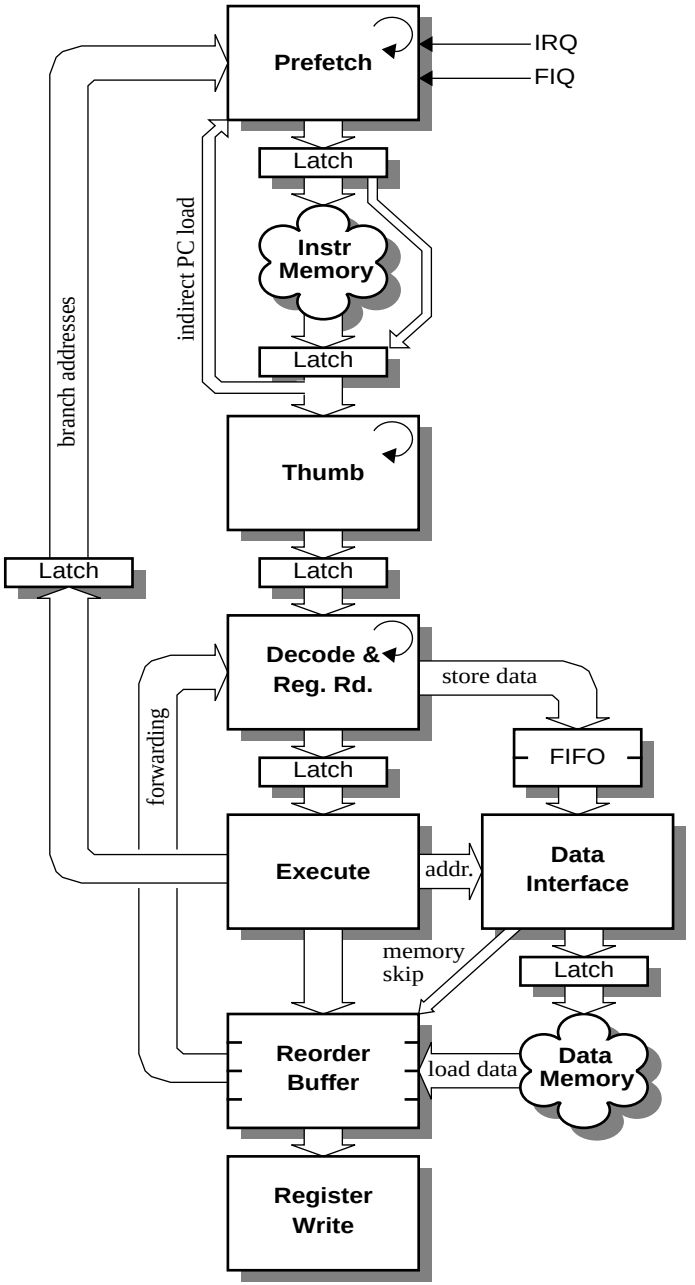
Working with the **ARM** architecture:

```
LDMFD   SP!, {R0-R3, PC}
```

LoaD Multiple, Full Descending stack the set of {R0, R1, R2, R3, PC} using base address SP and writing in back in the process (a big 'POP', used for procedure return)

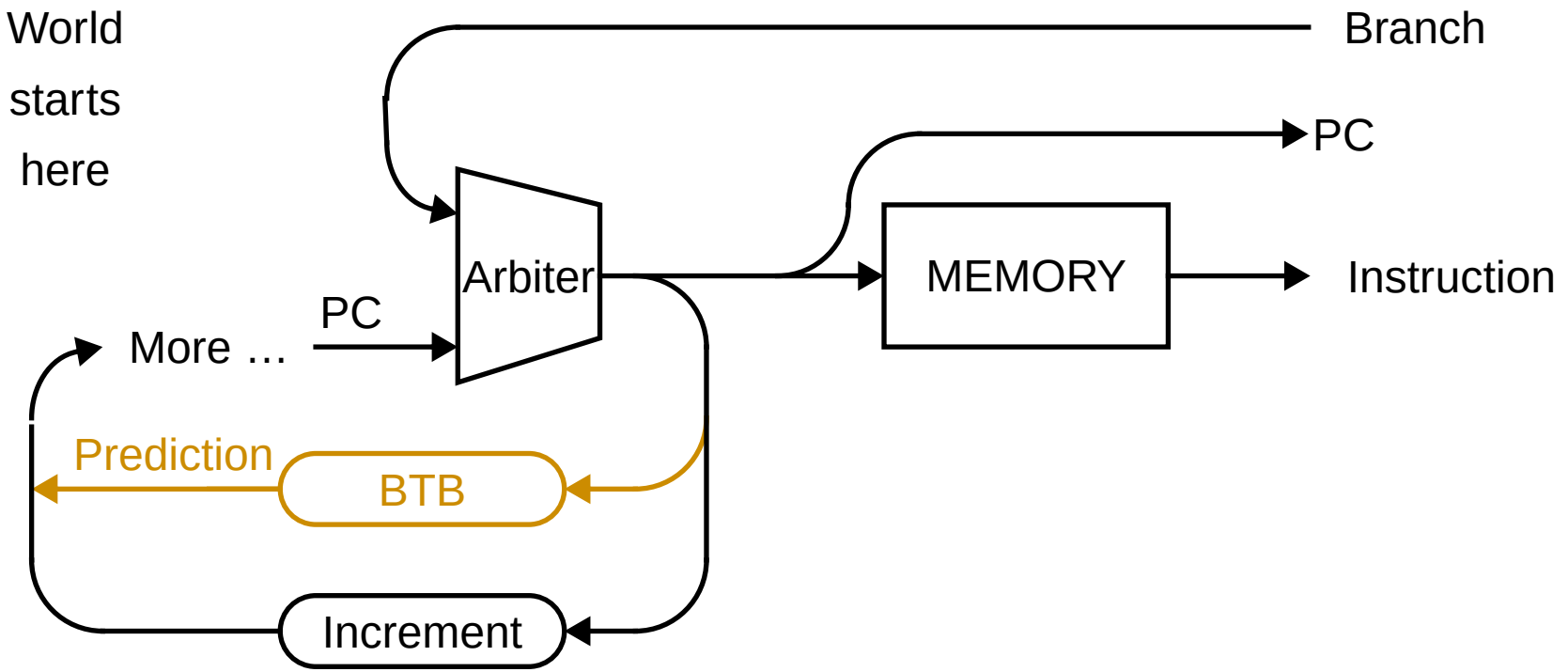
- Illustrates:
  - parallel memory/ALU operation
  - multi-cycle operation
  - possibilities for abort (on any cycle)
  - a branch (change of PC)

AMULET3 Processor



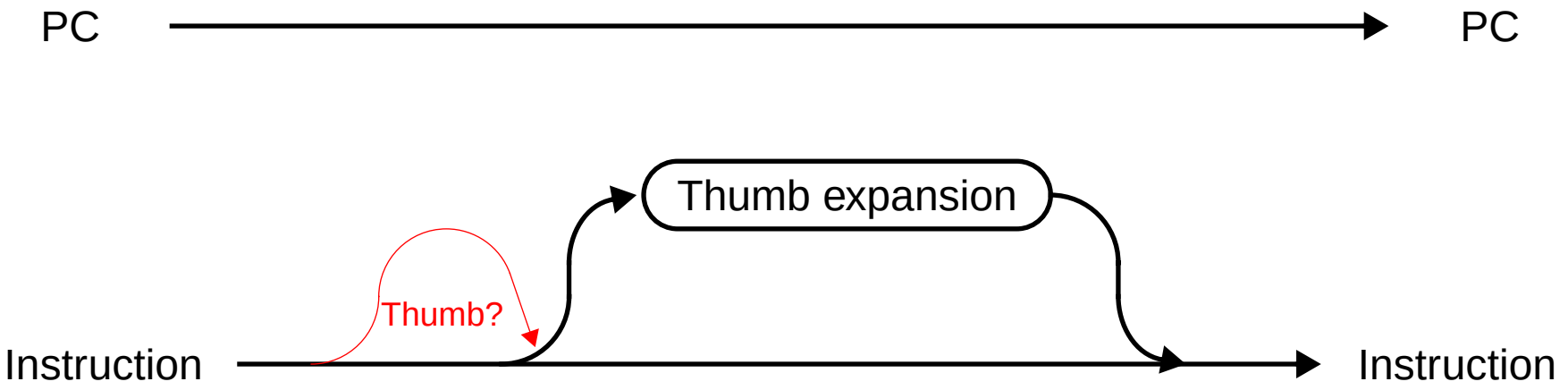
- Branch prediction
- Unwanted cycle suppression
- Automatic halt mode
- Thumb decoder
- Unrestricted register forwarding
- Load/store with out-of-order completion
- Dual (“Harvard”) bus interface
- Support for precise exceptions

Fetch stage



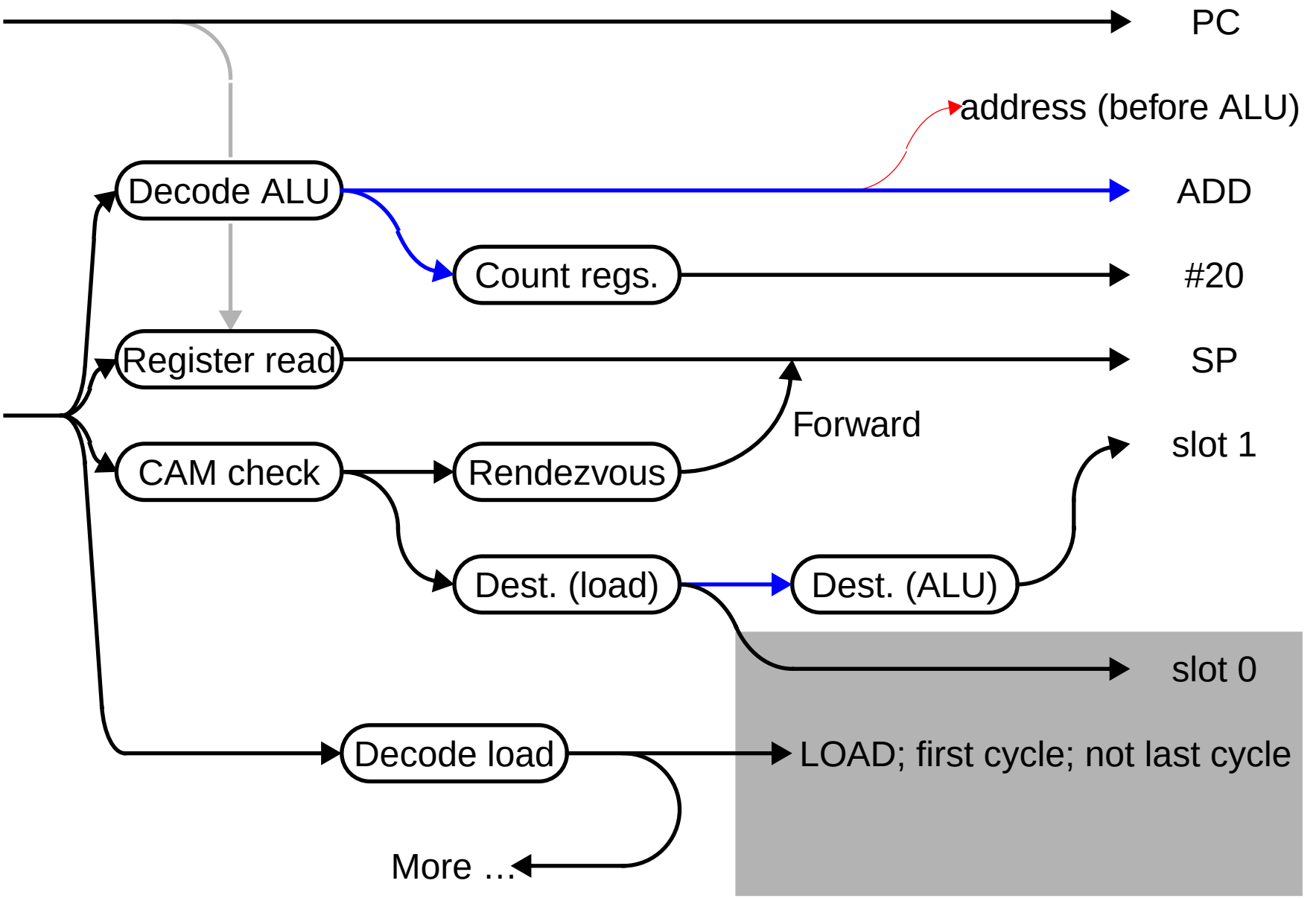
- ‘Free running’, incrementing loop
  - increment operation data-dependent – often very ‘cheap’
  - throttled if/when pipeline ‘backs up’
  - no information about pipeline structure considered
- Arbiter allows branches to occur
  - prefetch depth is *non-deterministic*

Precode stage

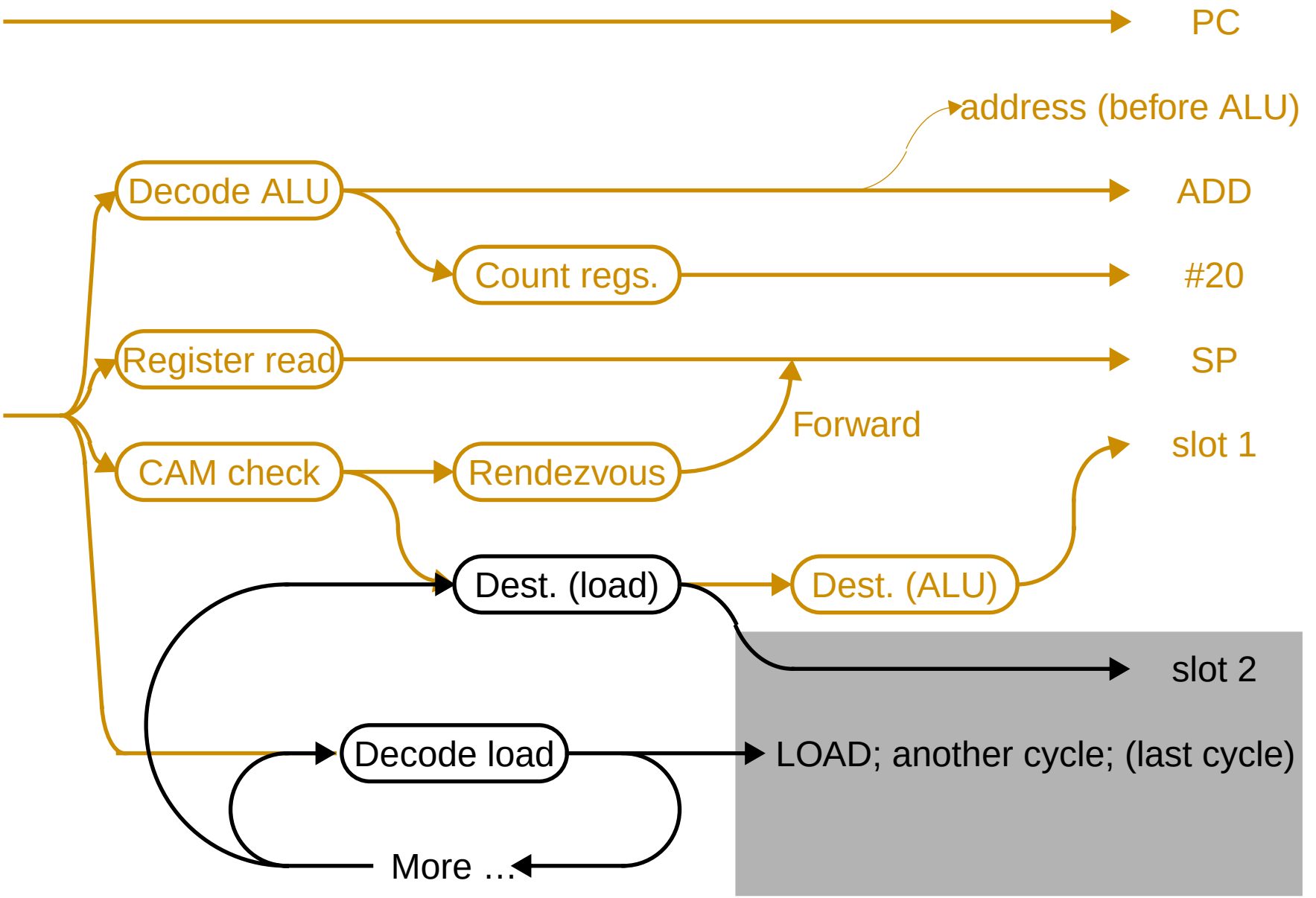


- Acts as a prefetch buffer
- May provide some decode
- Elastic pipeline stage
  - ARM instructions have low latency
  - Thumb instructions slower, when required
- Fetch address (PC) kept with instruction to avoid non-local interactions

Decode stage



Decoder stage



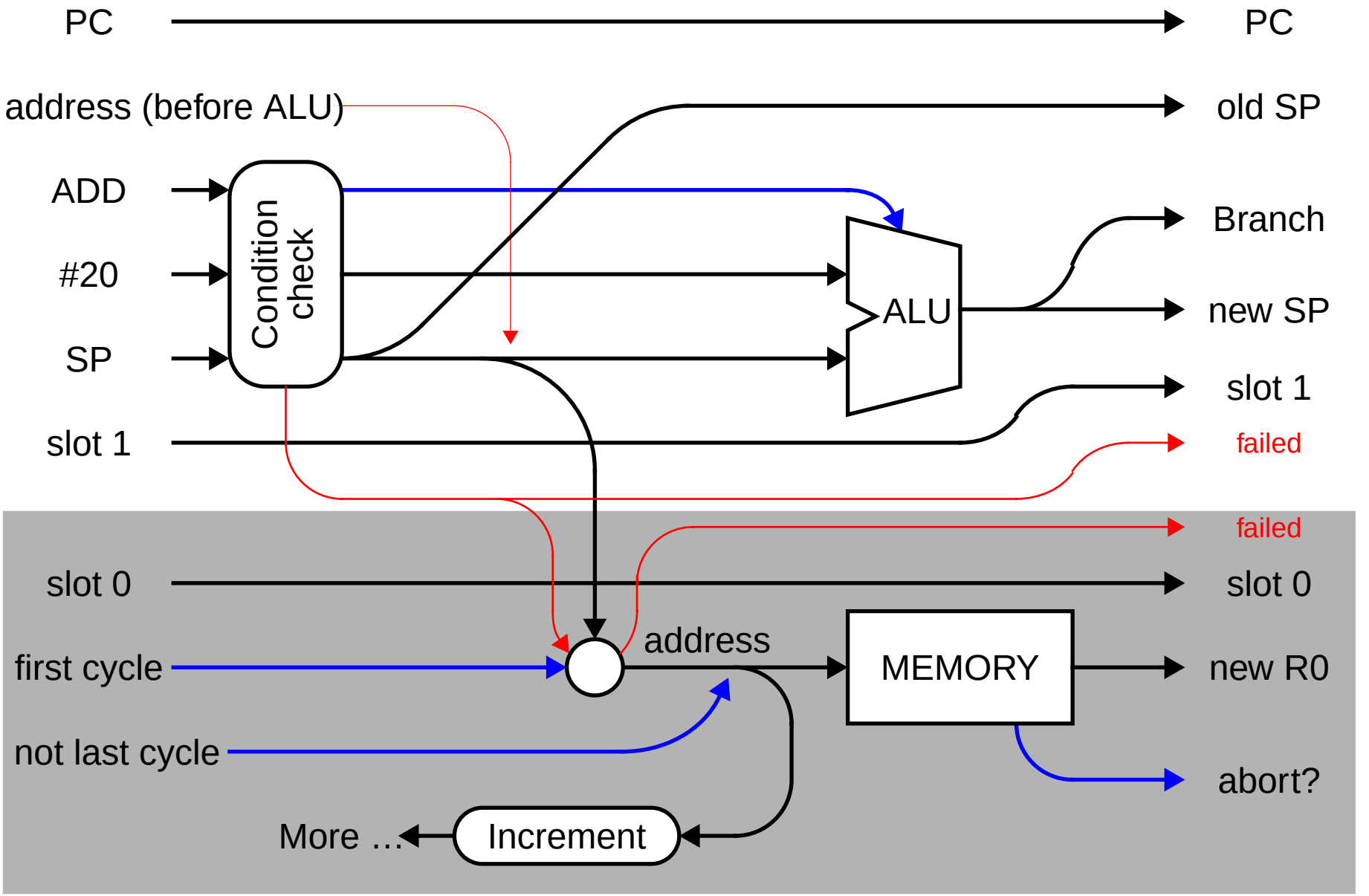
Decoder stage

- The register read process is triplicated
  - three read ports are a good match for ARM instructions
  - register identifiers are saved in a CAM; their location ‘slot’ in the CAM is carried with the parts of the instruction
  - a register *may* change during a read but, in all such cases, forwarding will correct the corrupted value

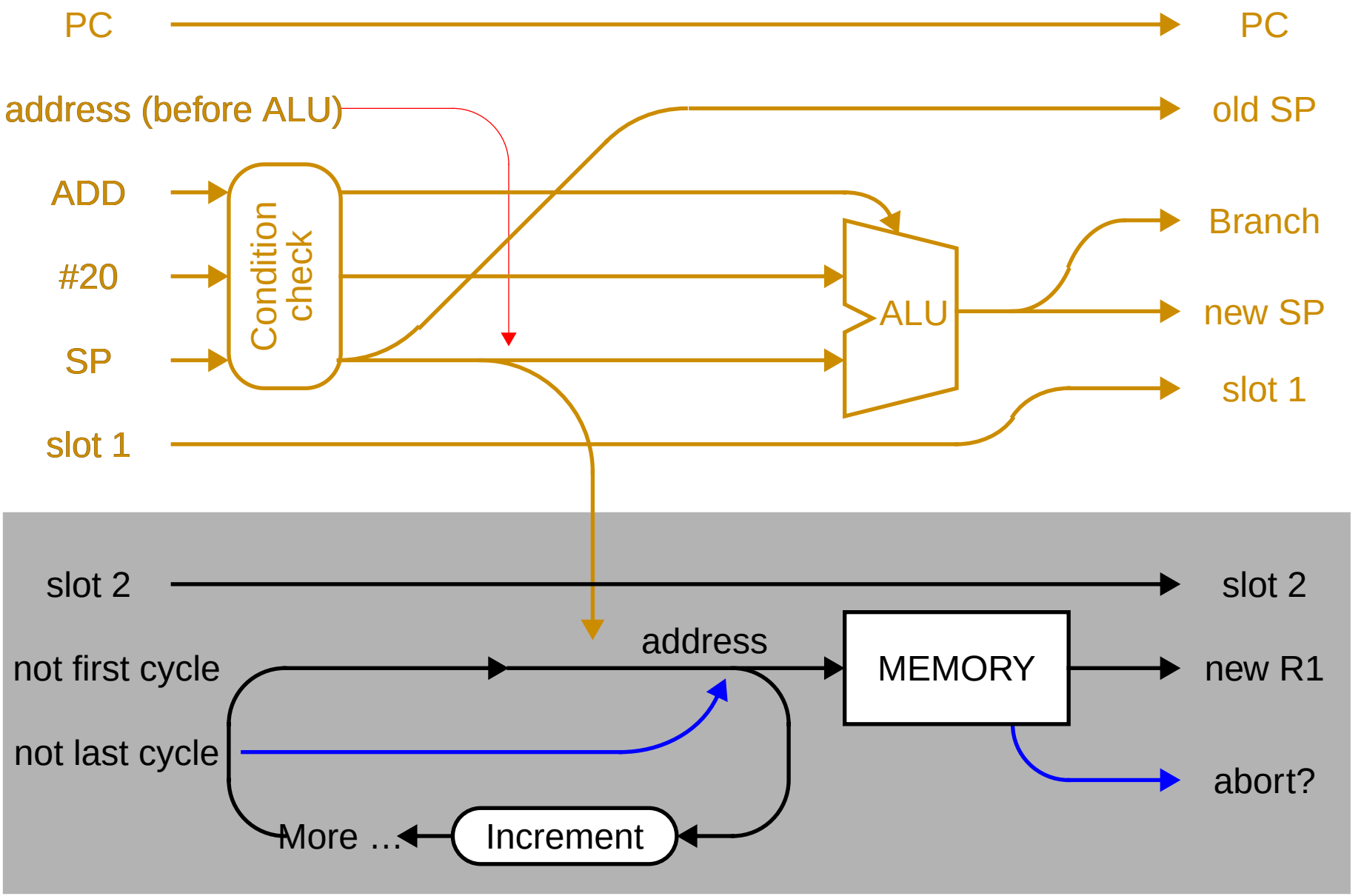
Asynchronous features

- ‘Thread level’ parallelism including variable delays
  - e.g. destination assignment: may be called 0, 1 or 2 times and may have to wait for a reorder buffer slot to be freed
- Multi-cycle operation
  - single instruction input causes multiple output cycles
  - no need for global control; fetching ‘backs up’ and stalls automatically

Execution stage



Execution stage



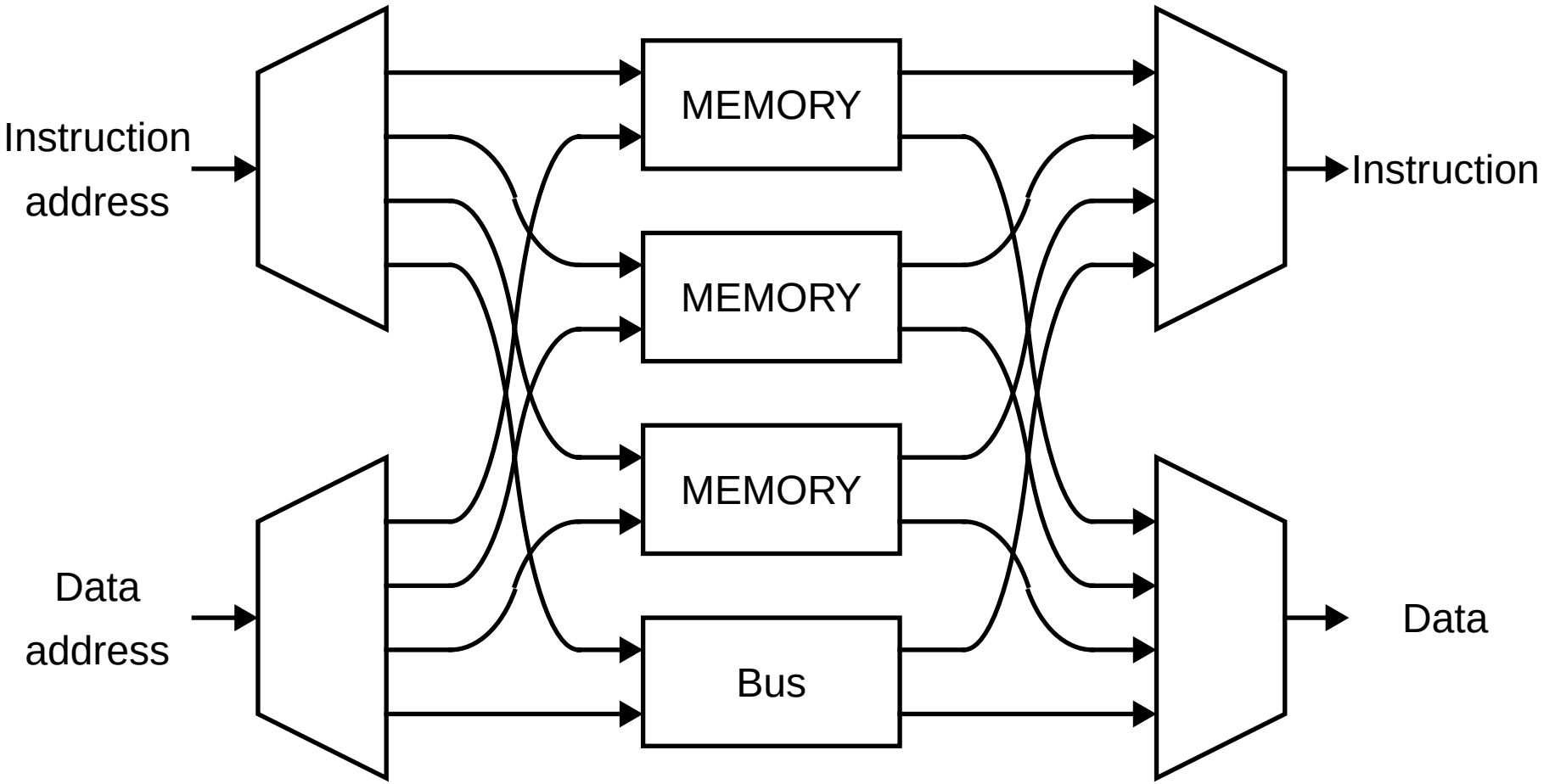
Execution stage

- Condition code flags are kept in this stage
  - similar mechanism used to discard wrongly prefetched instructions
- History buffer
  - for LDM and STM (only) the instruction's PC and the original value of the base register are saved, in case an abort occurs
  - this information is discarded on the final cycle of the instruction …
  - … or recovered and restored (via the reorder buffer) on an abort

Asynchronous features

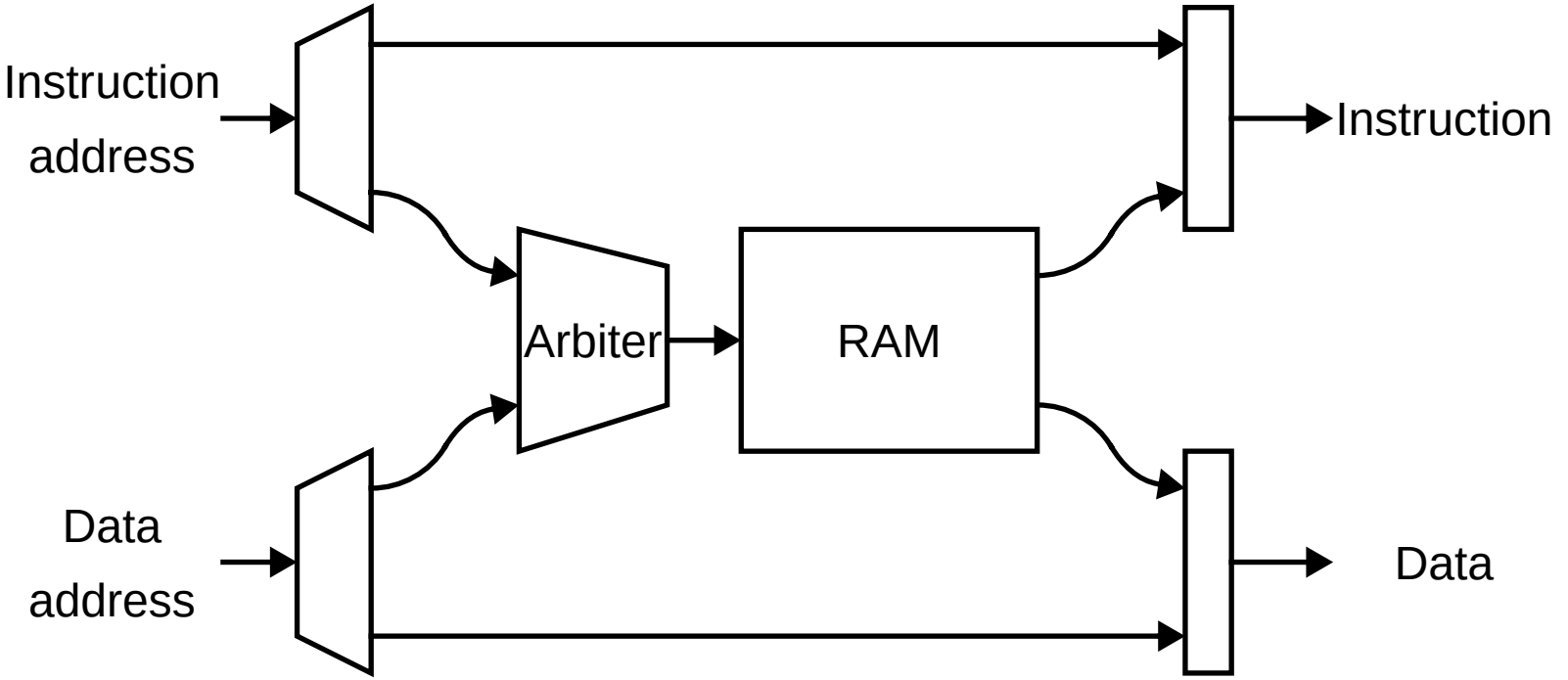
- Elastic – some operations can take longer than others
  - in particular 'failed' operations take less time
  - pre-indexed memory operations take longer (address is sourced after ALU)
- Automatic shutdown
  - the majority of the stage is only active during the first cycle of LDM
  - inactive circuits dissipate almost no power

Memory



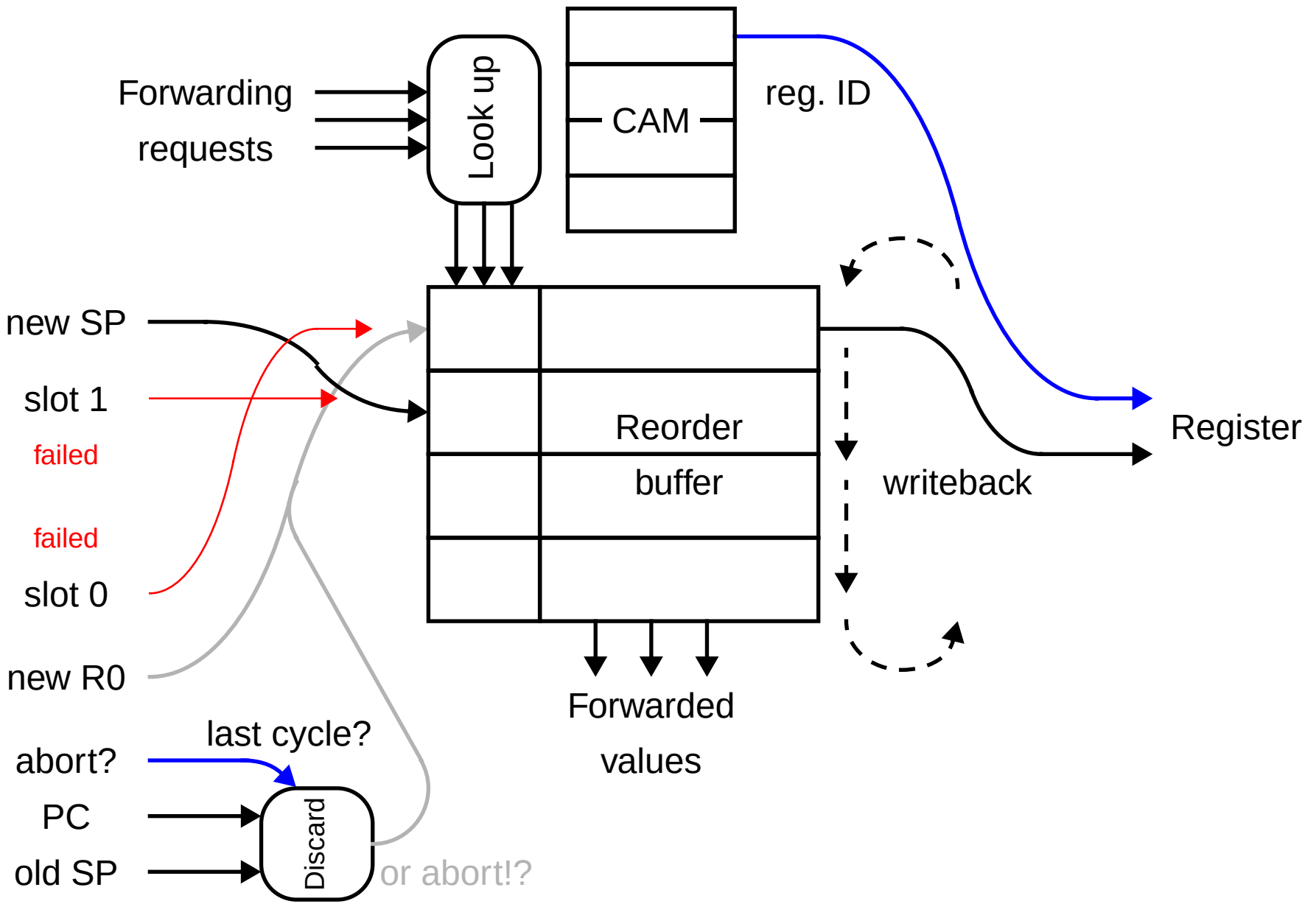
- Harvard bus on processor (buses cycle at different speeds)
- Local memory is banked to reduce probability of collision (8 banks + bus)
- Requests on different buses are fully asynchronous

Memory (detail)



- Many collisions are averted by a single line ‘cache’
- In case of genuine contention, an arbiter allocates the RAM on a ‘first come, first served’ basis
- A process may wait an indeterminate time for a previous access to finish
  - rare, and so what?
- Deadlock is avoided if process can always finish with protected resource

Reorder Buffer stage



Reorder Buffer stage

Note: even parts of the *same* instruction arrive here asynchronously!

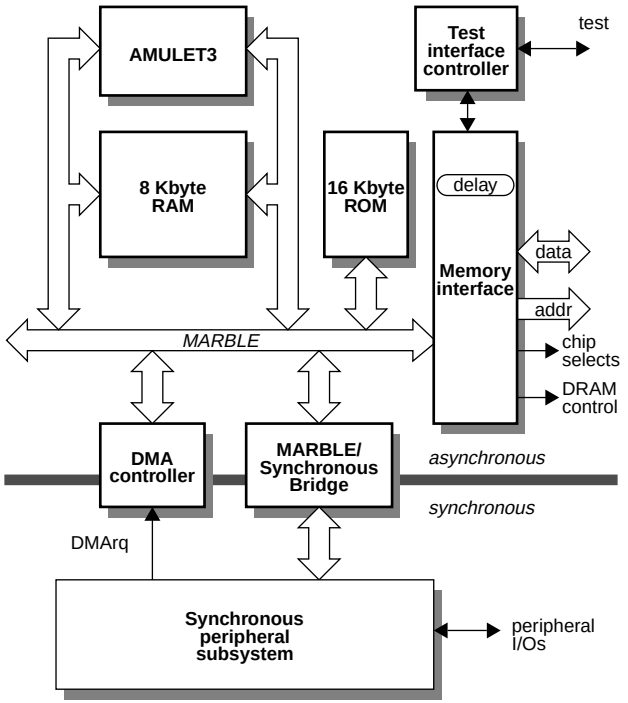
The key to the reorder buffer operation is the **two** validity flags.

- The ‘full’ bit
  - set when data arrives in a predesignated slot (out of order)
  - writeback process waits for *next* slot to be ‘full’ (i.e. in order), copies out data and clears ‘full bit’
- The ‘forward colour’ bit
  - *changes* when data arrives
  - remains in this state until changed by *next* data arrival
  - data remains valid for forwarding before, during and after writeback
- Forwarding knows the ‘expected’ colour and can check it at any time
- Providing it is predetermined (by the decoder) that a value will pass through a reorder buffer slot, it can then be checked asynchronously
  - may or may not be required to wait

MANCHESTER 1824

The University of Manchester

Amulet3i
- an Asynchronous System-on-Chip



- Amulet3 microprocessor (ARMv4T)
- 8 Kbytes RAM
- 16 Kbytes ROM
- Flexible multi-channel DMA controller
- Programmable external memory interface
- MARBLE, a fully asynchronous on-chip bus
- Bridge to on-chip synchronous bus
- Configuration registers
- Software debug support
- Test interface

Amulet3i – Vital Statistics

- Transistor count
  - AMULET3 – 113 000
  - RAM (total) – 504 000
  - DMA controller – 70 000
  - EMI – 26 000
  - Total – (asynchronous subsystem 800 000
- Area
  - AMULET3i – ~25mm^2
  - AMULET3 – ~3mm^2

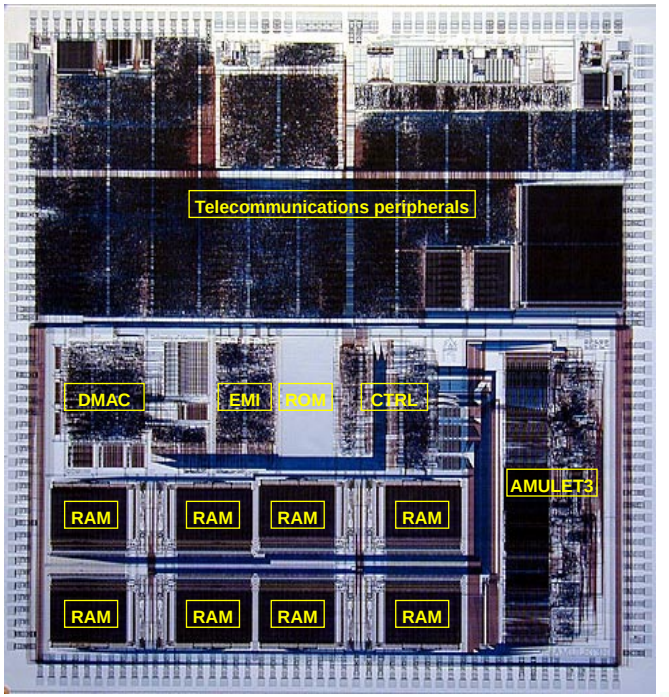
Note: the local RAMs are relatively large in these generic, ASIC rules.

- System Performance (Measured)
  - Peak Native MIPS 79 MIPS (96 in Thumb code)
  - 149 kDhrystones/s – 85 Dhrystone MIPS (ARM)
  - 108 kDhrystones/s – 62 Dhrystone MIPS (Thumb) (-30%)
- AMULET3i power average 130 mW
  - 60% is within the processor core (simulation result)

Comments

- AMULET3i is about 2x faster than AMULET2e
  - The speed-up is about 1.4x when normalised for the different processes (0.35µm vs. 0.5µm)
- Async. vs. sync.
  - 0.35µm Amulet3 ⇒96 MHz, (103 Dhrystone MIPS) 1100 MIPS/W
  - 0.35µm ARM9 ⇒120 MHz, (133 Dhrystone MIPS) 800 MIPS/W
  - EMI considerably lower in Amulet3

DRACO:
DECT Radio Communications Controller



Conclusions on asynchronous processor design

- Designing an asynchronous processor is different from a synchronous one
  - some assumptions are not valid
  - need to think of *processes* rather than cycles
- Particularly nice bits
  - halting and local stalling
  - easy dynamic power management
  - EMC
- Some things can be done in similar fashion to synchronous design …
- … other things need a new way of thinking, e.g.
  - forwarding
  - non-determinism – deadlocks are a danger
  - exploiting variable (sometimes data-dependent) timing
- It's possible to be (at least) competitive with synchronous design