ECE 455 – Module 1

Review of Embedded Computing

Spring 2023

Murray Dunne – mdunne@uwaterloo.ca

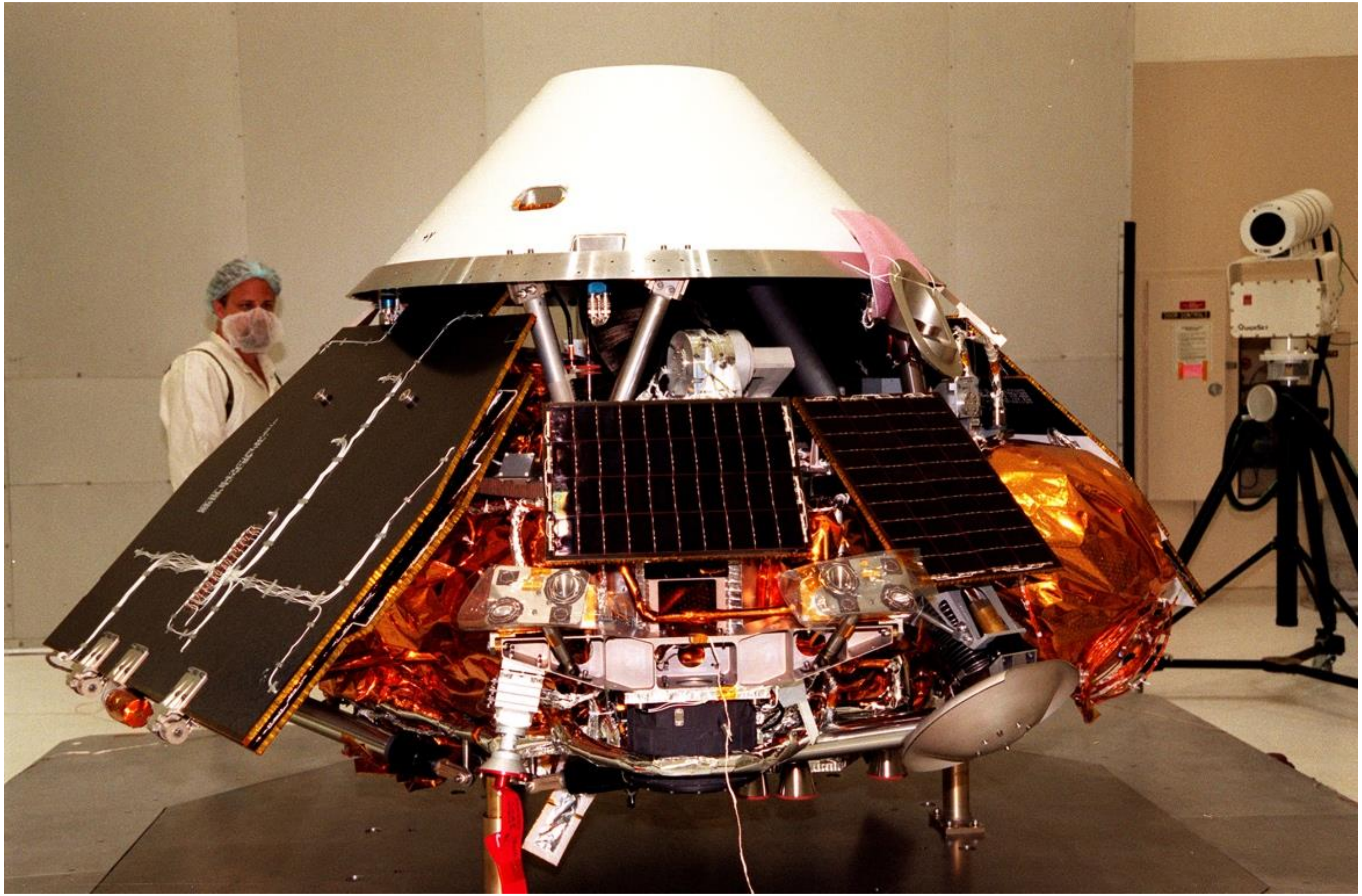
Slides Acknowledgement

Some material in these slides is based on slides from:

- Prof. Sebastian Fischmeister
- Prof. Carlos Moreno

Other material used with citations

Disaster of the Day – Mars Polar Lander (1999)



What is an **Embedded System**?

- A system of hardware and software built for a specific purpose

- Vs. a general-purpose computer
  - Not built for a specific purpose

- Vs. a hardware only solution
  - Missing software component: nothing to be “embedded”

Why use a microcontroller instead of:

- A purely hardware circuit solution?
- A standard desktop/laptop computer?

Some Challenges

- How much hardware is necessary?
- How to ensure we meet the deadlines?
- How to minimize power consumption and resource usage?
- How to design for longevity and upgradability?
- Will it really work and be safe and secure?

Requirements

- **Functional requirement**: how the system must behave
  - Example: when you flip the switch, the light turns on
- **Non-functional requirement**: any other requirements
  - Example: each light switch should cost less than $5
  - Types of non-functional requirements include:
    - Performance
    - Cost
    - Physical weight
    - Size
    - Power consumption

Testing

- **Model-in-the-loop**: you have a model of the system that you can test against a simulated environment
- **Software-in-the-loop**: you can run the software against a simulated environment
- **Hardware-in-the-loop**: you can run the software on the hardware against a simulated or real environment
- **System-in-the-loop**: you can run everything against a real environment

System Integration

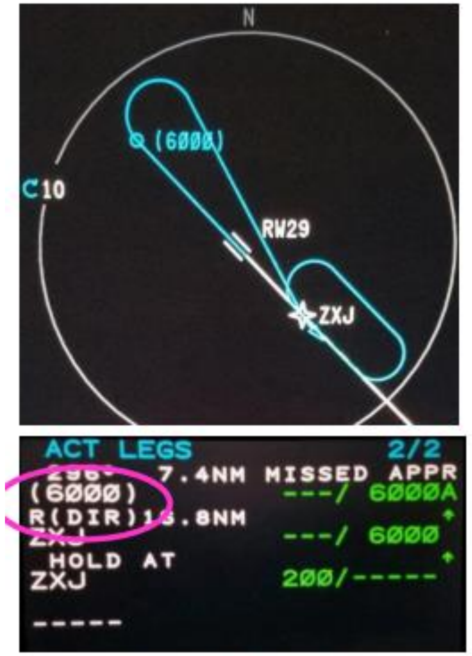
- Is really hard!
  - Real-world system users will encounter infinitely more possibilities than you can test
  - Reproducibility
    - Hidden timing constraints
  - Hardware issues manifest as software issues
    - Example: loose connector causes corrupted packet leads to crash
      - Is this the software’s fault?

Validation vs. Verification

- **Validation**: did we build the right system?
  - Example: customer ordered a plane, we built a car
- **Verification**: did we build the system right?
  - Example: customer ordered a plane, but our plane crashes all the time

Disaster of the Day
Bombardier CRJ-200 Rockwell Collins FMS (2017)

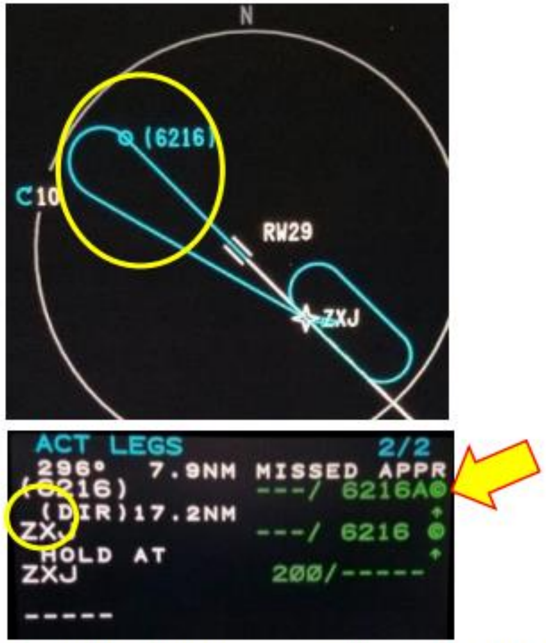
CC BY-SA 2.0 - formulanone - flickr



```
ACT LEGS                 2/2
 296°   7.4NM MISSED APPR
(6000)           ---/ 6000A
R(DIR)16.8NM
ZXJ              ---/ 6000
 HOLD AT
ZXJ              200/-----

-----
```

Figure 2a. "Climb to 6000 feet followed immediate by Right turn Direct-to ZXJ.



```
ACT LEGS                 2/2
 296°   7.9NM MISSED APPR
(6216)           ---/ 6216A
 (DIR)17.2NM
ZXJ              ---/ 6216
 HOLD AT
ZXJ              200/-----

-----
```

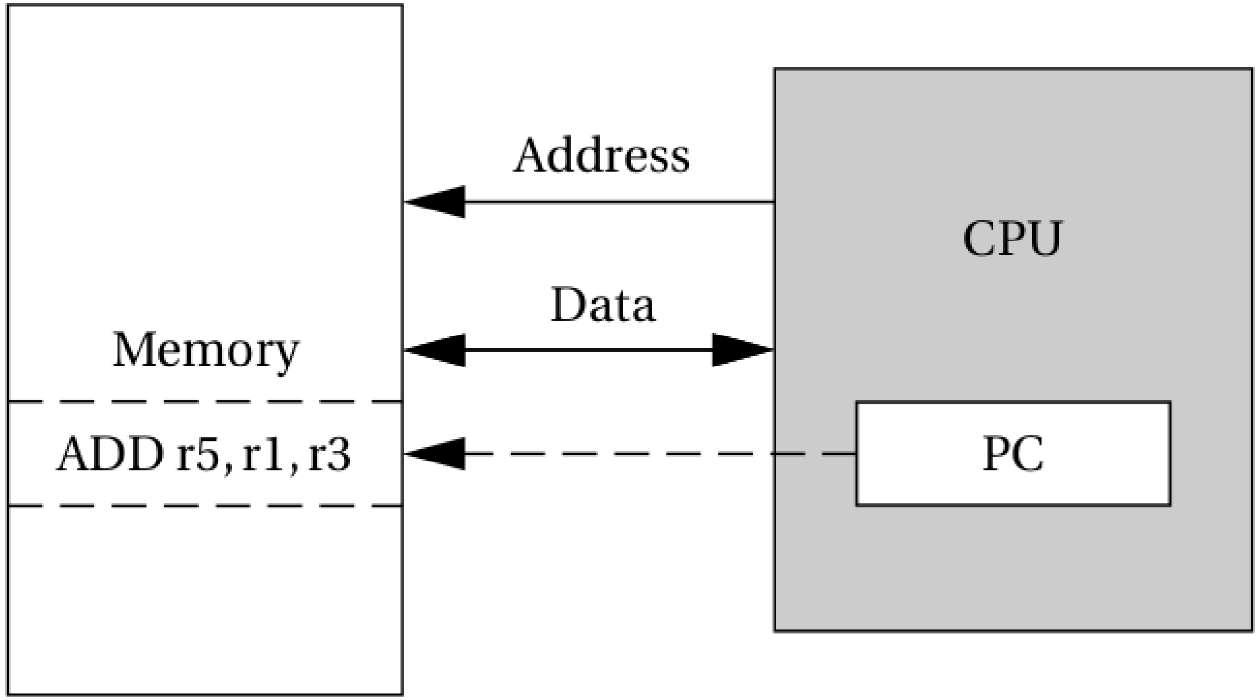
Figure 2b. After the crew uses the left FMS to activate Temperature Compensation, the "R" turn to ZXJ is initially removed (blanked) on the right FMS and will subsequently be removed on both FMSs. The map display may differ from this example.

From: Rockwell Collins OPSB 0166-17R4

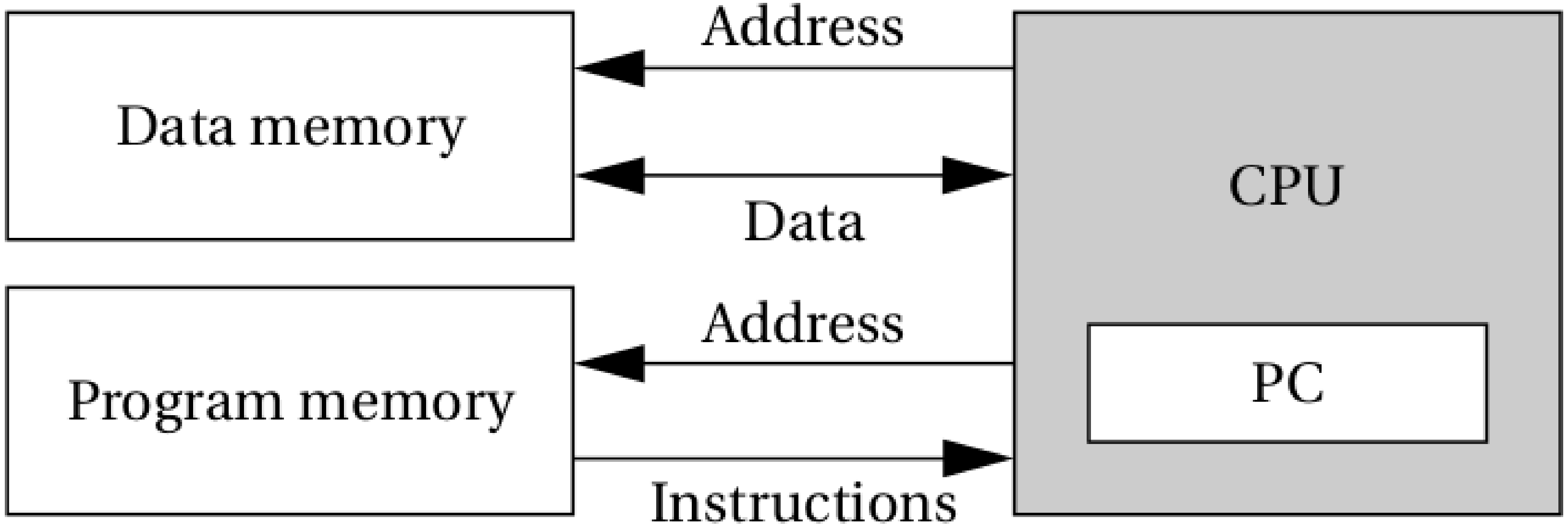
Computer Architecture Terms Review

- **Central Processing Unit (CPU)**
- **Memory**
- **Registers**
- **Program Counter**
- **Program Status Register**
- **CISC vs. RISC**

Computer Architectures – von Neumann



Computer Architectures – Harvard



Computer Architectures - Comparison

- von Neumann
    - More efficient use of available memory

- Harvard
    - Parallelization of memory and data access
    - Protection from code injection
    - Harder to write self-modifying programs

Assembly Language

- One step above machine code
  - Textual representation of machine code
- One instruction per line
  - First column for labels
  - Second column for instruction and operands

- Example:

```
add r0, r3, #5
```

Machine Language

- Underlying binary representation of instructions
- **ARM**: popular Instruction Set Architecture (ISA)
  - Used in the labs in this course
    - Arm Cortex M3 specifically
    - https://developer.arm.com/documentation/dui0552/a/the-cortex-m3-instruction-set/instruction-set-summary?lang=en
  - Very popular in industry

Machine Language

31	27	25	24	20	19	15	11	0
cond	00	X	opcode	S	Rn	Rd	Format determined by X bit	

X = 1 (represents operand 2):

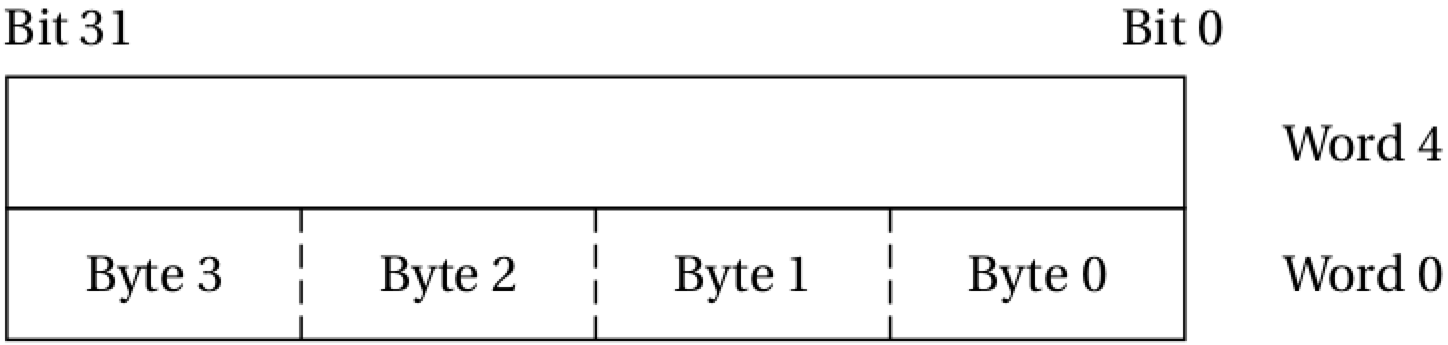
11	7	0
#rot	8-bit immediate	

X = 0 format:

11	6	4	3	0
#shift	Sh	0	Rm	

11	7	6	4	3	0
Rs	0	Sh	1	Rm	

Endianness

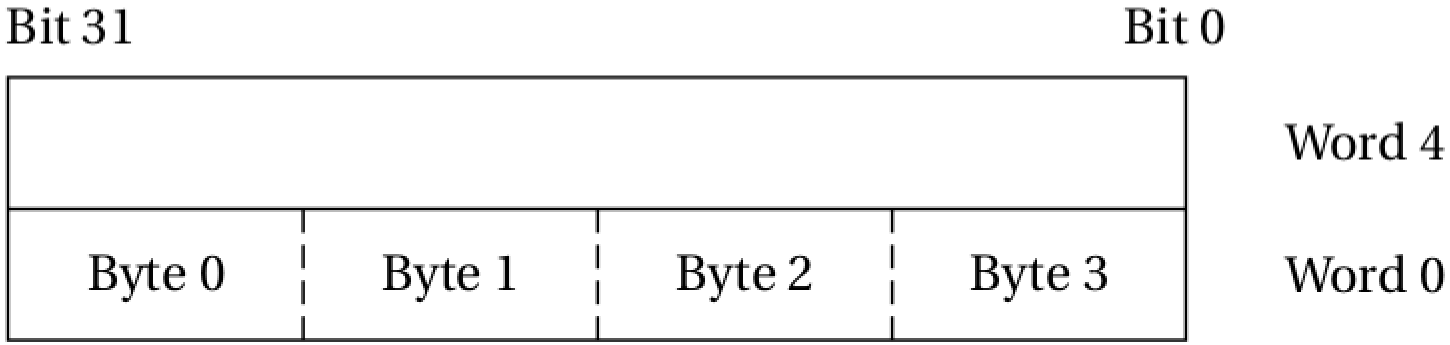


Bit 31 Bit 0

Word 4

Byte 3 Byte 2 Byte 1 Byte 0 Word 0

Little-endian



Bit 31 Bit 0

Word 4

Byte 0 Byte 1 Byte 2 Byte 3 Word 0

Big-endian

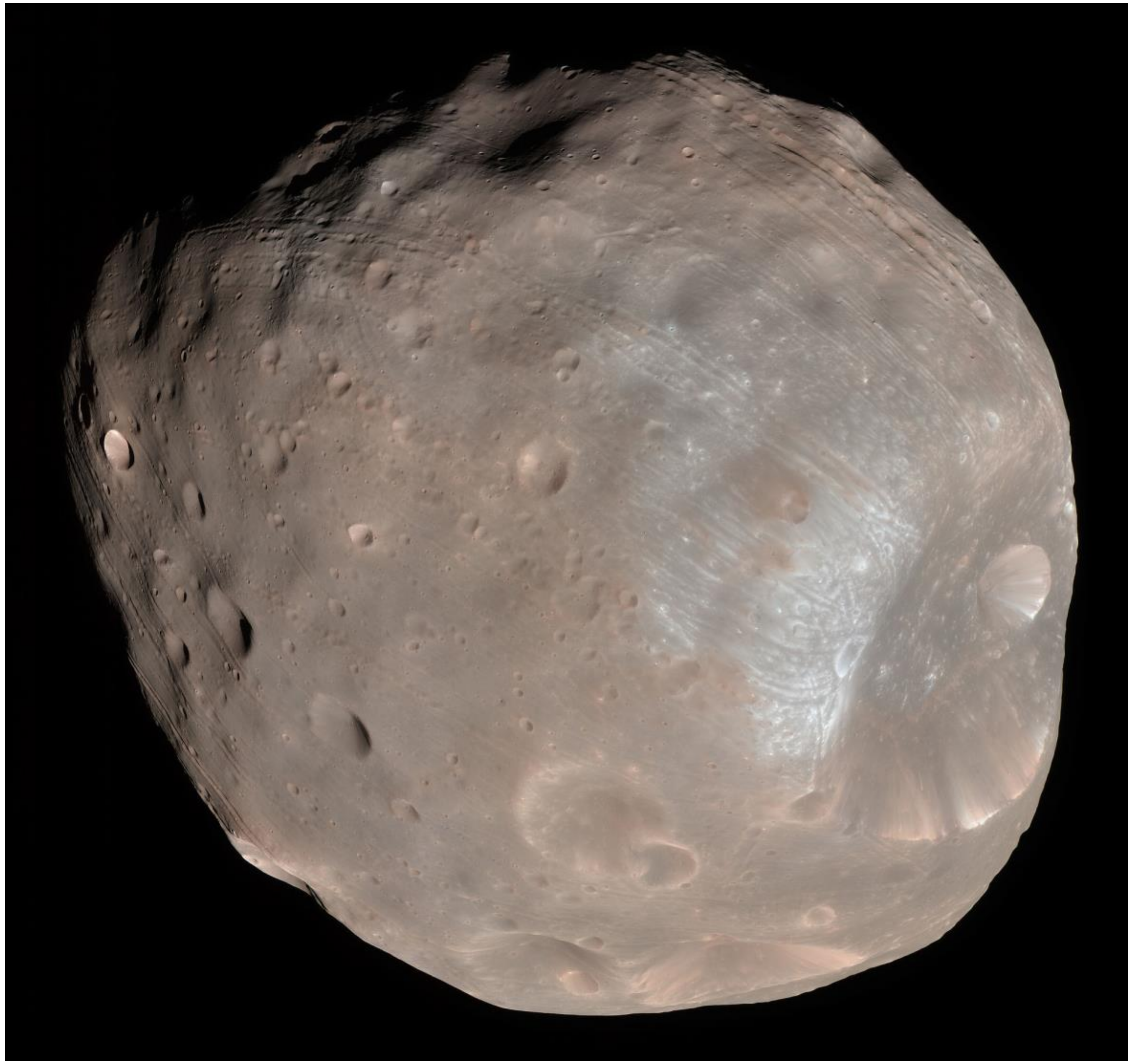
Core Registers

- **Program Counter** (PC)
  - Where are we in the program?
- **Stack Pointer** (SP)
  - Where are we in memory?
- **Program Status Register** (PSR)
  - What is going on right now?
  - Arm Cortex M3 has three PSRs
- R0-R12
  - 32-bit general purpose registers

Stack Pointer vs. Frame Pointer

- Stack Pointer (SP)
    - Where are we in memory *right now?*
- **Frame Pointer** (FP)
    - Where are the local variables of the current function?
    - Usually **`r11`** in ARM

Disaster of the Day – Phobos 1 (1988)



State Machine vs. Threads

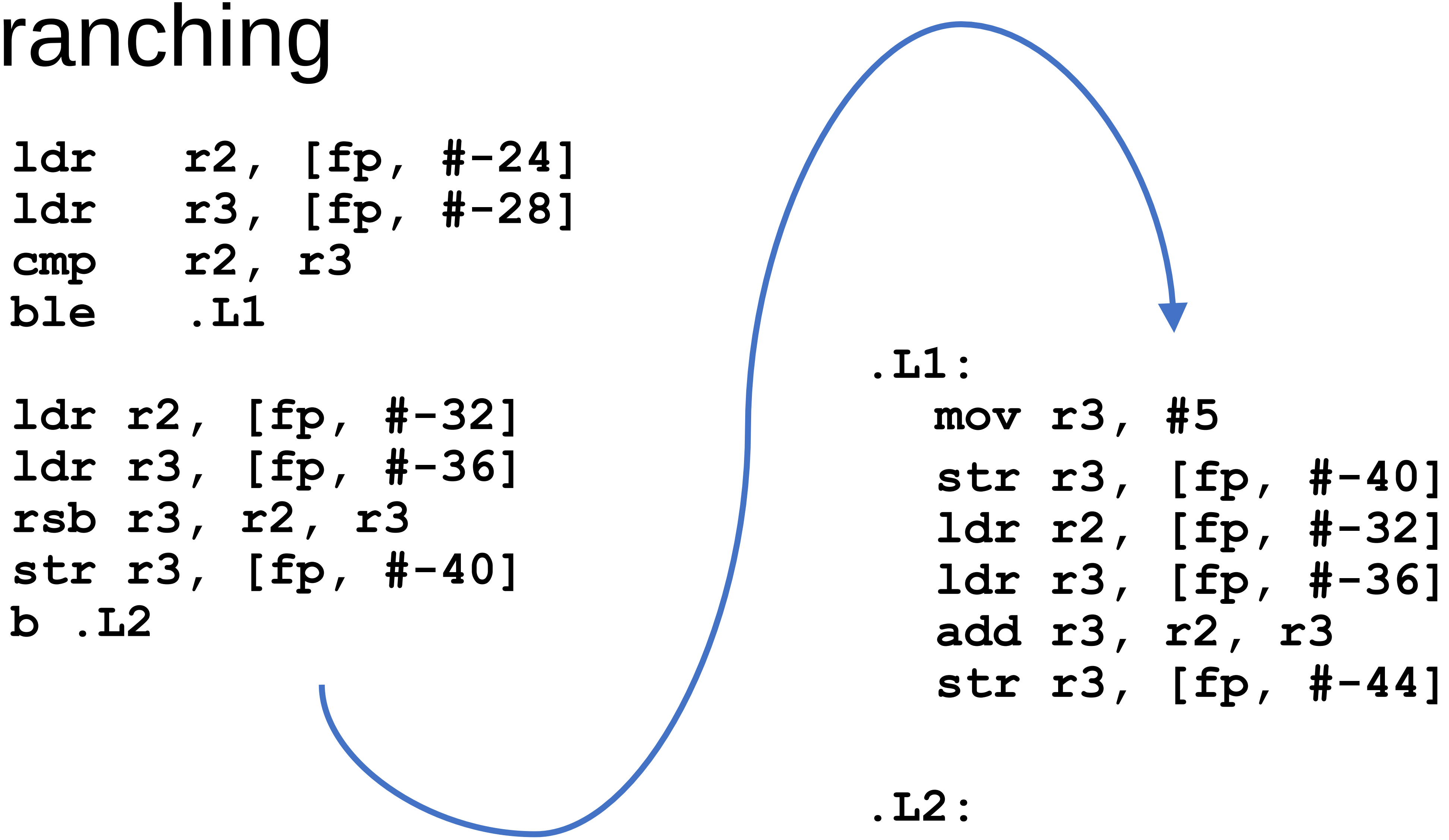
- **State machine**
  - Formal language construct
  - Execution transitions between states

- **Threads** (or **Processes**)
  - Well defined, but not a formal language
  - Support concurrency
    - Not actually that relevant for embedded systems
  - Simplify representation of underlying state machine
    - Useful once you need more than a handful of states

Addressing Modes

- ARM instructions have different addressing modes
  - `mov r0, r1`
    - **Register direct**
  - `add r0, r1, #3`
    - **Immediate** (also called Literal)
  - `ldr r0, [r1]`
    - **Register indirect**
  - `ldr r0, [r1, #4]`
    - **Register indirect with offset**
  - `ldr r0, [PC, #4]`
    - **PC relative** (also **PC indirect with offset**)

Branching



```
ldr   r2, [fp, #-24]
ldr   r3, [fp, #-28]
cmp   r2, r3
ble   .L1

ldr r2, [fp, #-32]
ldr r3, [fp, #-36]
rsb r3, r2, r3
str r3, [fp, #-40]
b .L2
```

```
.L1:
  mov r3, #5
  str r3, [fp, #-40]
  ldr r2, [fp, #-32]
  ldr r3, [fp, #-36]
  add r3, r2, r3
  str r3, [fp, #-44]

.L2:
```

I/O Access

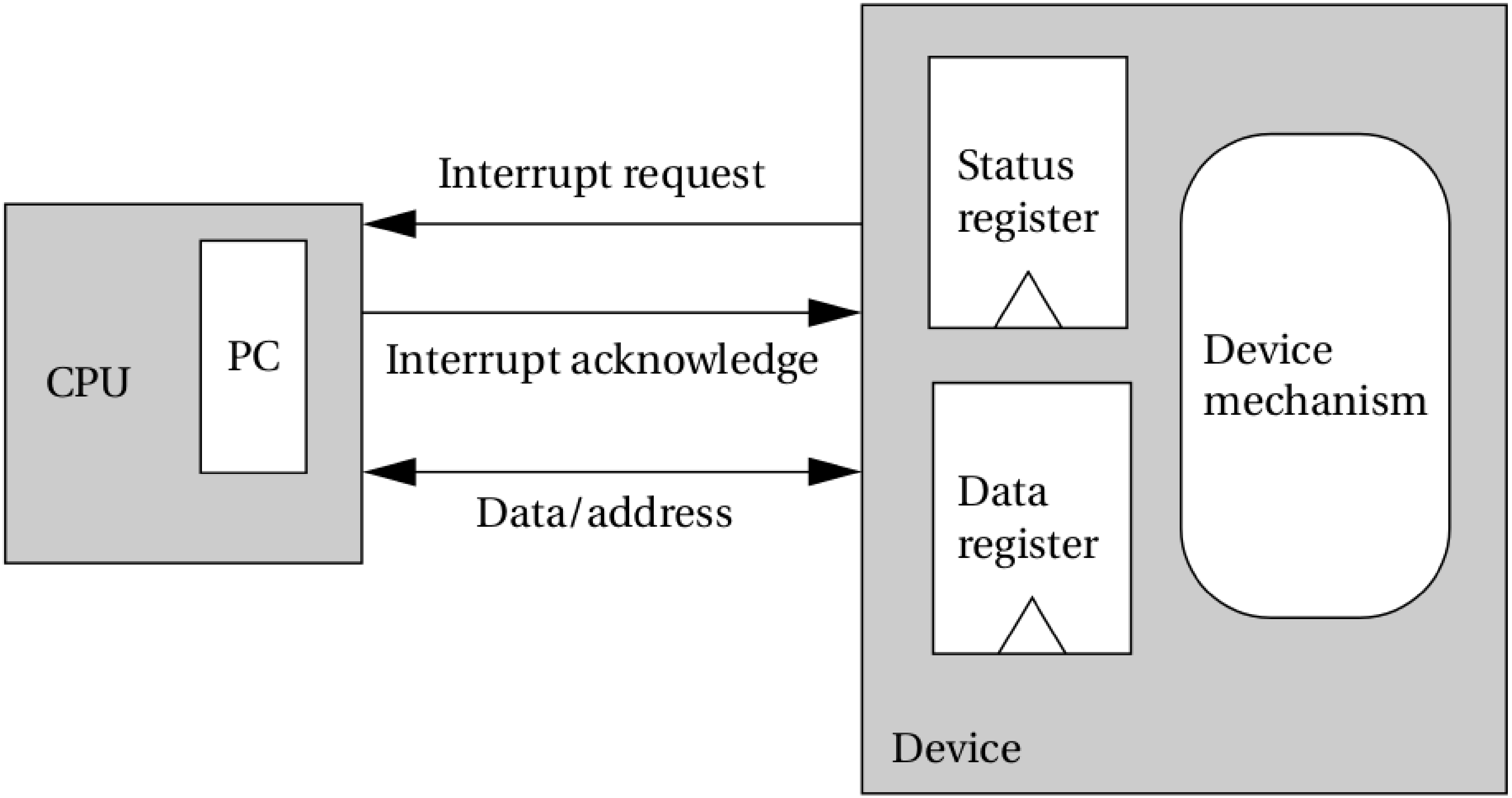
- **I/O Instructions**
  - Specific instruction to read from or write to a device
  - Example:
    - **`out r0, #AF45`**

- **Memory Mapped I/O**
  - Memory locations “connected” to external device
  - Much more popular

Polling vs. Interrupts

- In **Polling** we repeatedly check for inputs at predefined times
- **Interrupts** allow is to handle data when notified externally
  - More popular than polling, but not a panacea
    - What are some downsides of interrupts?

Interrupts



Interrupt request
CPU
PC
Interrupt acknowledge
Data/address
Status register
Data register
Device mechanism
Device

Interrupts

- We have to clear the interrupt
  - What if there are multiple queued?
  - Does the peripheral support multiple?
- Overhead
  - Lookup handler in vector table
  - Subroutine call
  - Branch penalties
    - Pipeline stalls, branch mispredictions
  - Saves current PC
    - Perhaps other state?
  - Post interrupt state restore

Interrupt Priority

- Multiple interrupts could occur (nearly) simultaneously
  - How to we prioritize?
- **Interrupt Vector Table**
  - Has reference to all handlers
  - Enforces priority
    - Called NVIC in ARM
- **Non-maskable Interrupts (NMI)**
  - Highest priority
  - Illegal instructions, power failure

Interrupt Process

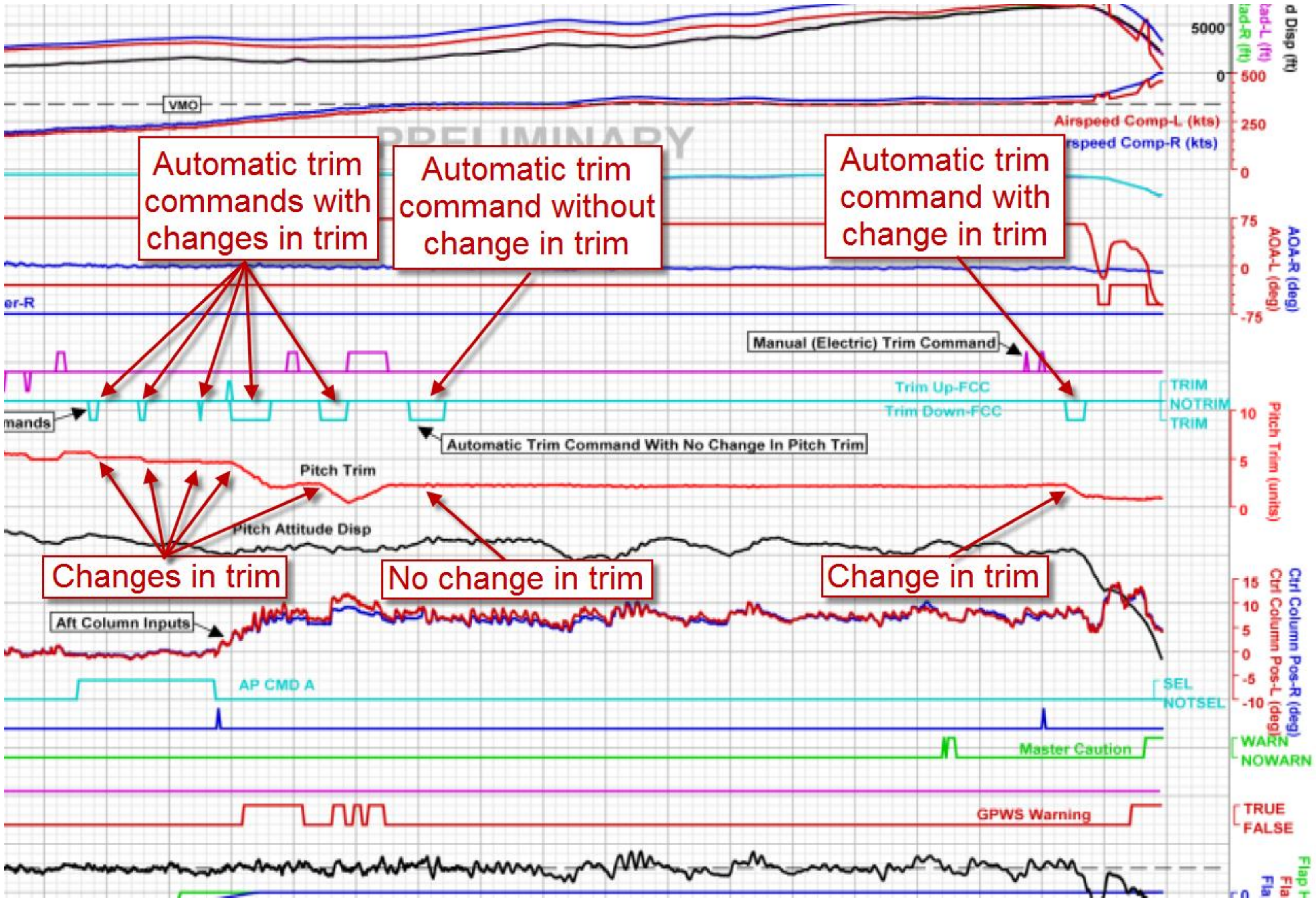
- Interrupt line raised
- CPU looks up handler address in Interrupt Vector Table
- Software saves state
- Handler executes
- Special return instruction restores state
  - **`refi`** in ARM

Modes

- Types of execution context
  - **User mode**
  - **Fast interrupt**
  - **Interrupt**
  - **Supervisor**
  - **Abort**
  - **System**
  - **Undefined instruction**
- Add hardware-supported safety to avoid programming errors

Disaster of the Day – Boeing 737 Max 8 (2018)

Disaster of the Day – Boeing 737 Max 8 (2018)



Automatic trim commands with changes in trim

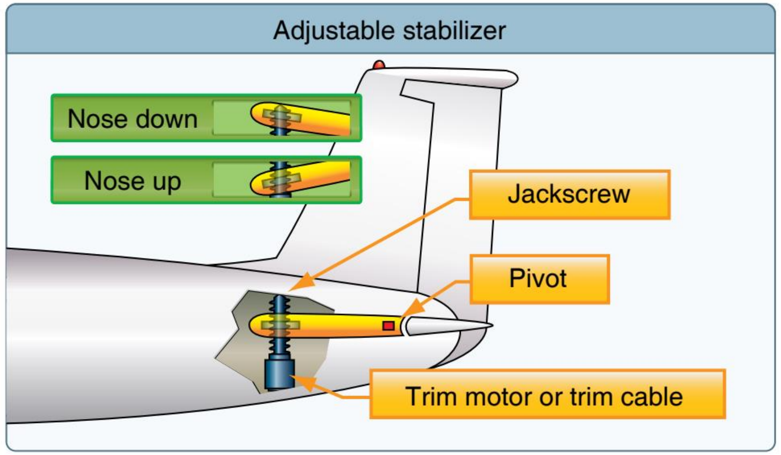
Automatic trim command without change in trim

Automatic trim command with change in trim

Changes in trim

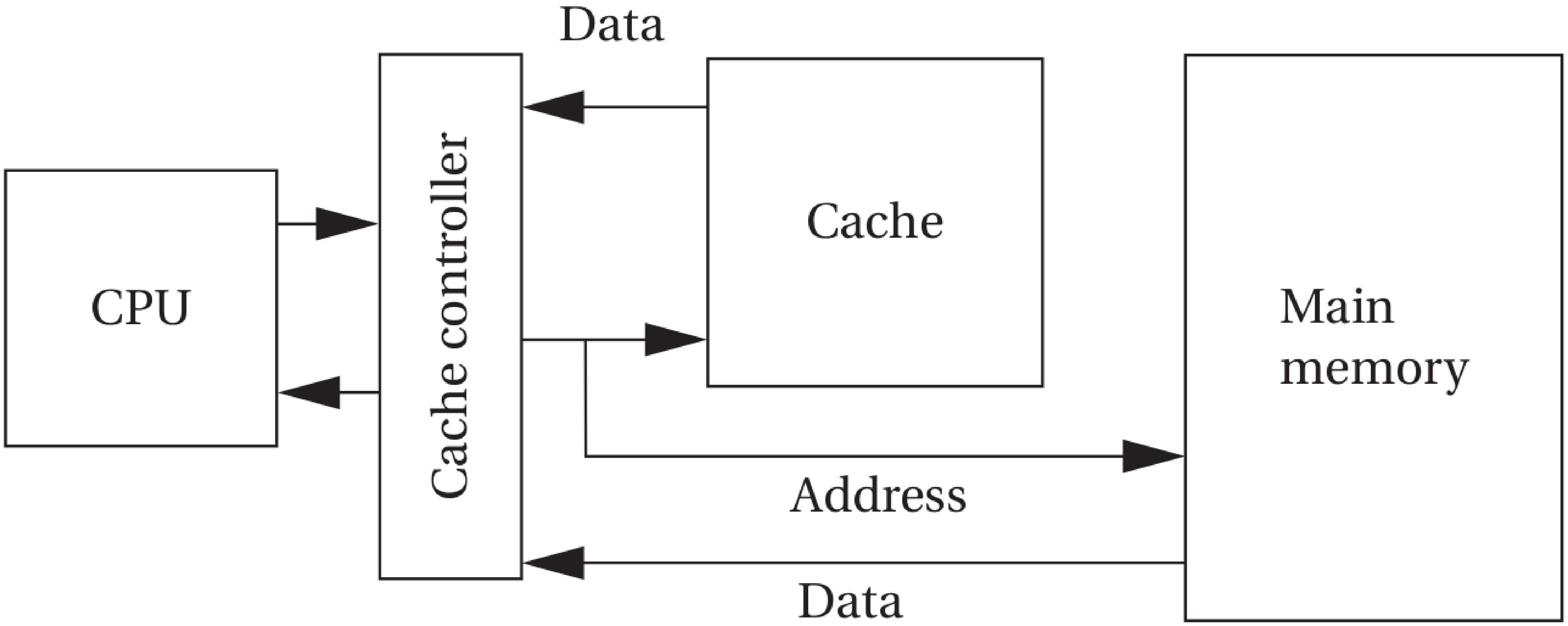
No change in trim

Change in trim



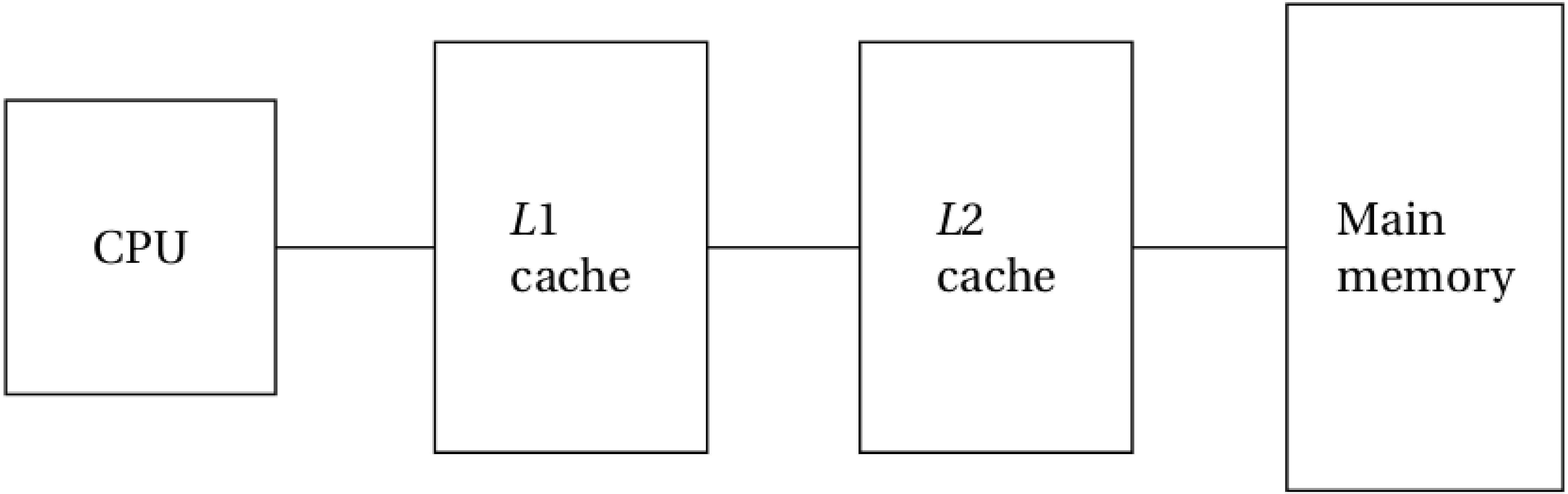
FAA U.S. Department of Transportation
FEDERAL AVIATION ADMINISTRATION
Flight Standards Service
Pilot's Handbook of Aeronautical
Knowledge Ch 6

Caching



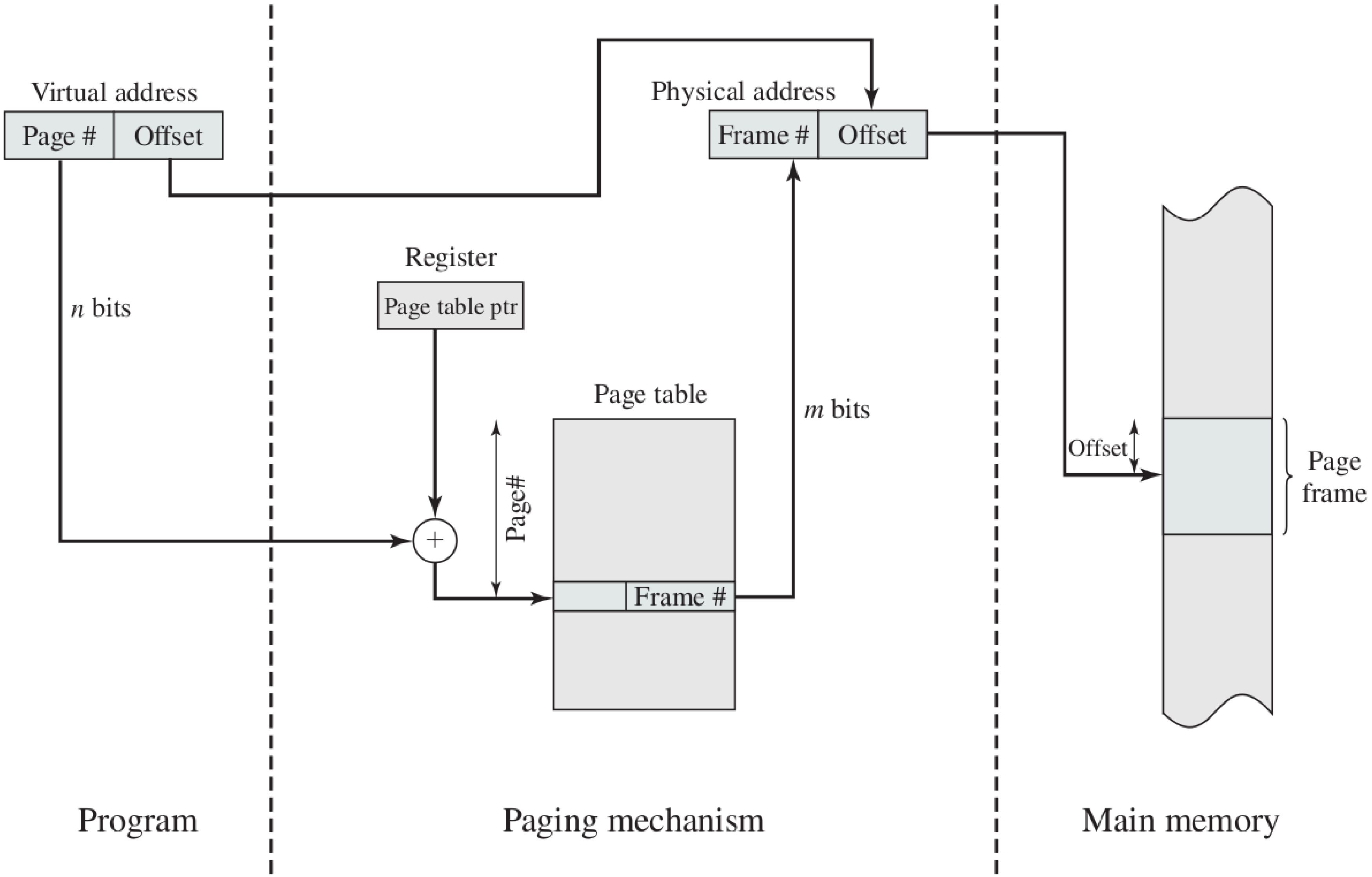
Average Memory Access Time (AMAT) = hit rate * access time + (1 – hit rate) * main memory access time

Caching

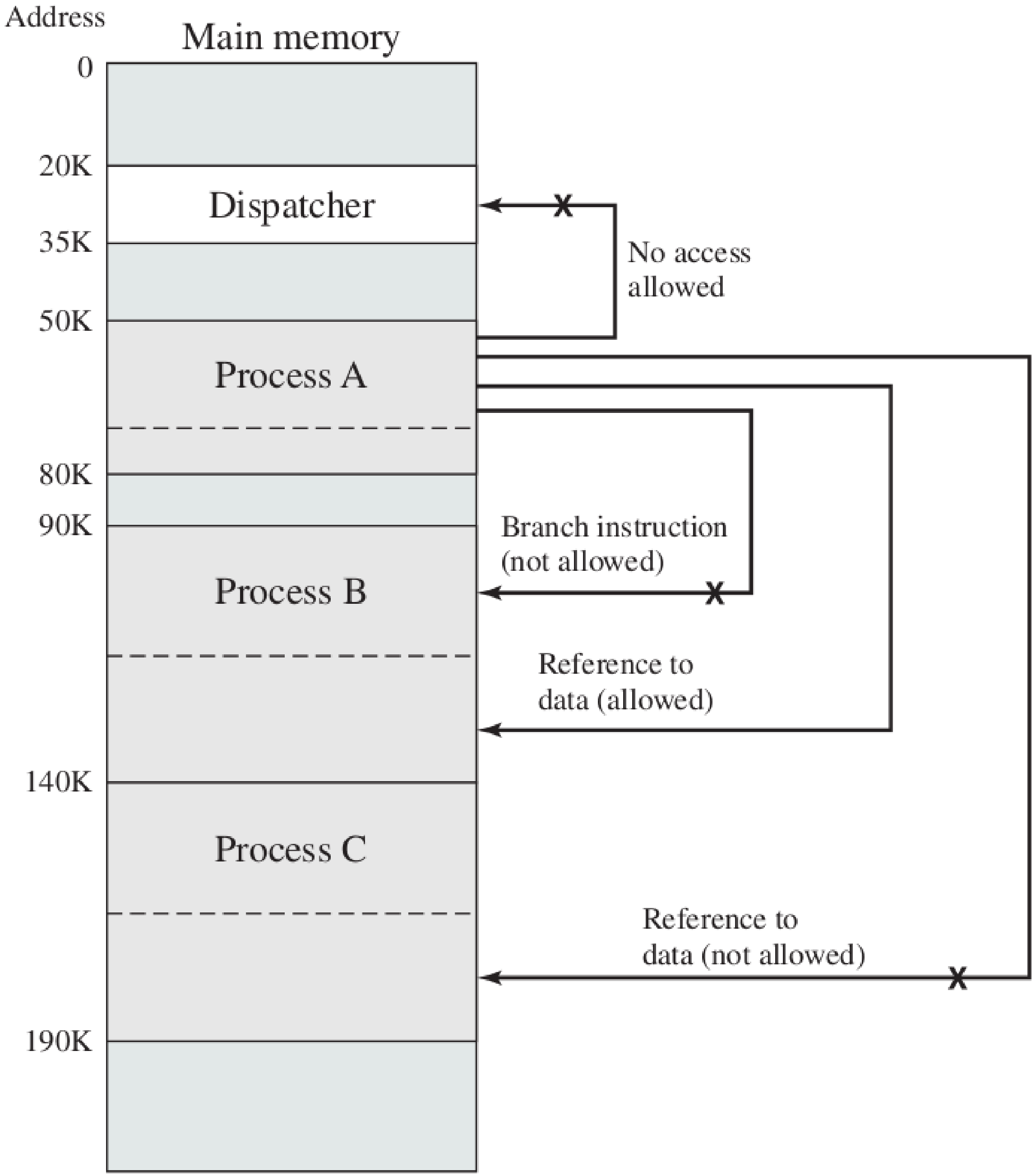


AMAT = L1 hit rate * L1 access time + (L2 hit rate – L1 hit rate) * L2 access time + (1 – L2 hit rate) * main memory access time

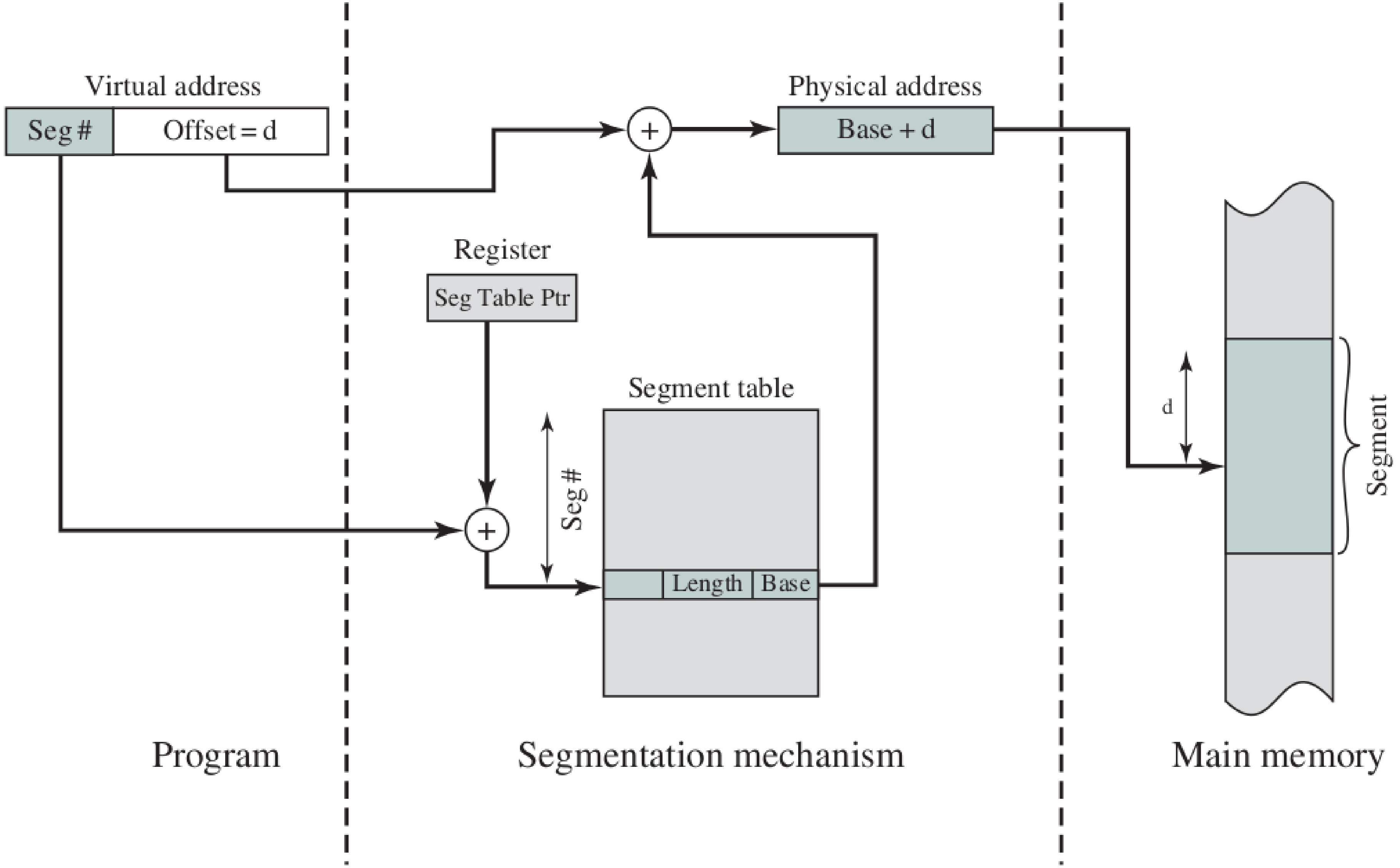
Paging


Virtual address
Page #
Offset
n bits
Register
Page table ptr
+
Page#
Page table
Frame #
Physical address
Frame #
Offset
m bits
Offset
Page frame
Program
Paging mechanism
Main memory

Virtual Memory

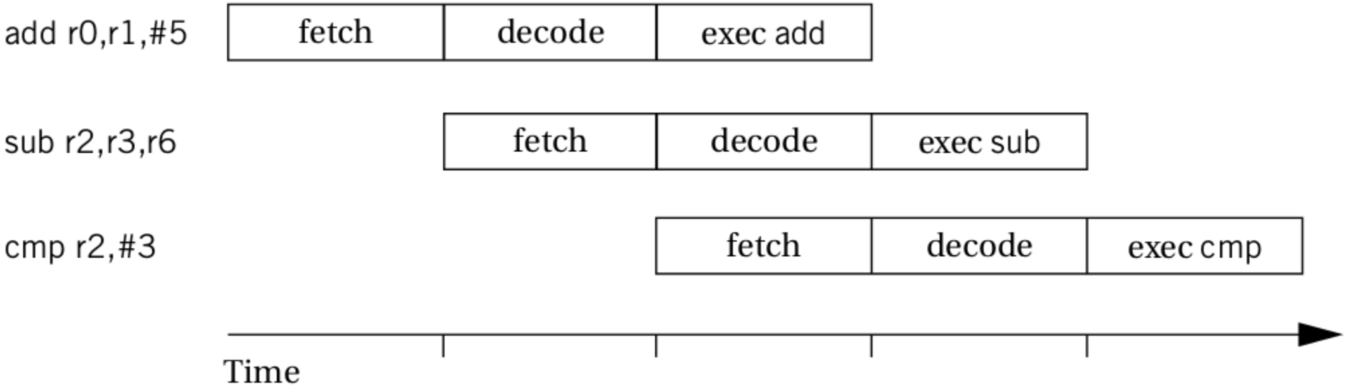


Segmentation



Processor Pipeline

- ARM pipeline has three stages – 3 cycles per instruction
  - Fetch
  - Decode
  - Execute
- Running them sequentially is inefficient
  - Overlap them instead
  - One cycle per instruction?



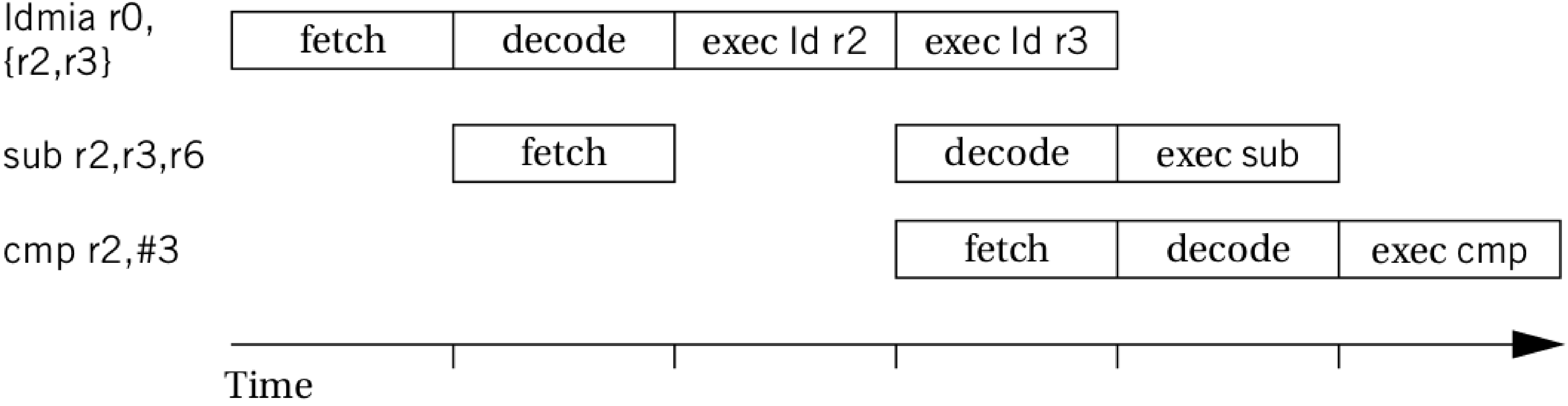
add r0,r1,#5 fetch decode exec add

sub r2,r3,r6 fetch decode exec sub

cmp r2,#3 fetch decode exec cmp

Time

Data Stall



ldmia r0, {r2,r3}

fetch | decode | exec ld r2 | exec ld r3

sub r2,r3,r6

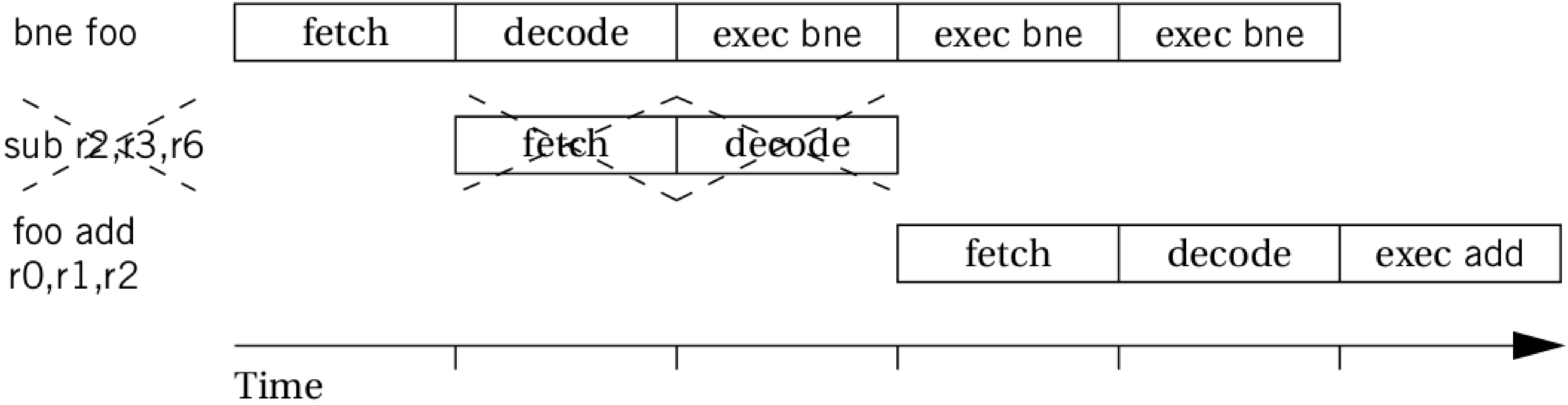
fetch | decode | exec sub

cmp r2,#3

fetch | decode | exec cmp

Time

Control Stall


bne foo
fetch
decode
exec bne
exec bne
exec bne
sub r2,r3,r6
fetch
decode
foo add r0,r1,r2
fetch
decode
exec add
Time