Embedded System CPUs: ARM7, Cortex M3

COE718: Embedded Systems Design

http://www.ee.ryerson.ca/~courses/coe718/

Dr. Gul N. Khan
http://www.ee.ryerson.ca/~gnkhan
Electrical and Computer Engineering
Ryerson University

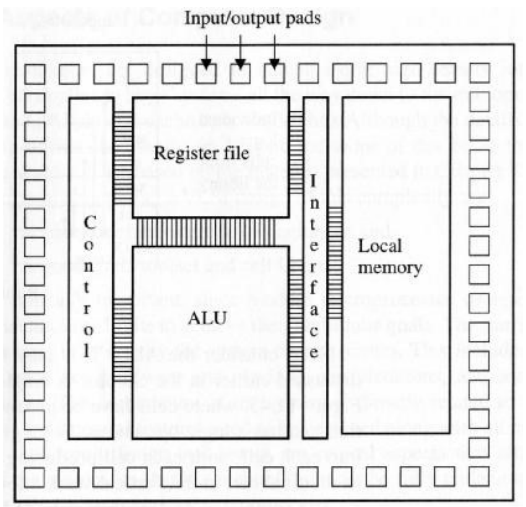
Overview

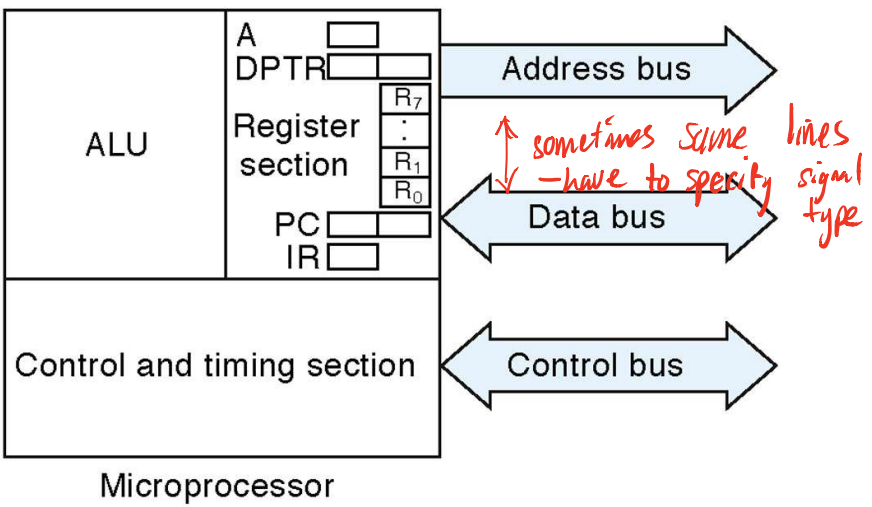
- Processors and System Architecture, Interrupts, Memory System
- Pipelining, I/O and CPU Performance
- Micro-controllers and Embedded CPUs
- ARM Architectures: ARM7TDMI
- Cortex-M3 a small foot-print Microcontroller

Text by Lewis: Chapter 5, part of Chapters 6 and 8 (8.1) and ARM7/M3 Data Sheets
Text by M. Wolf: part of Chapters/Sections 2.1, 2.2, 2.3 and 3.1-3.5

CPU/Processor Architecture

- Control and Timing Section
- Register Section
- ALU (Arithmetic Logic Unit)



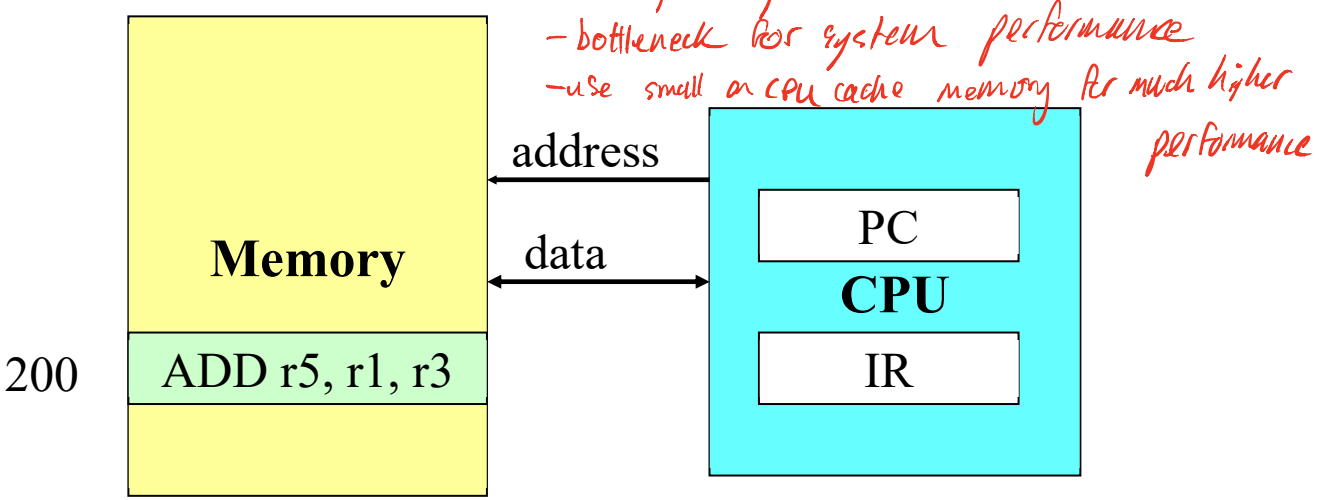


Processor Architectures

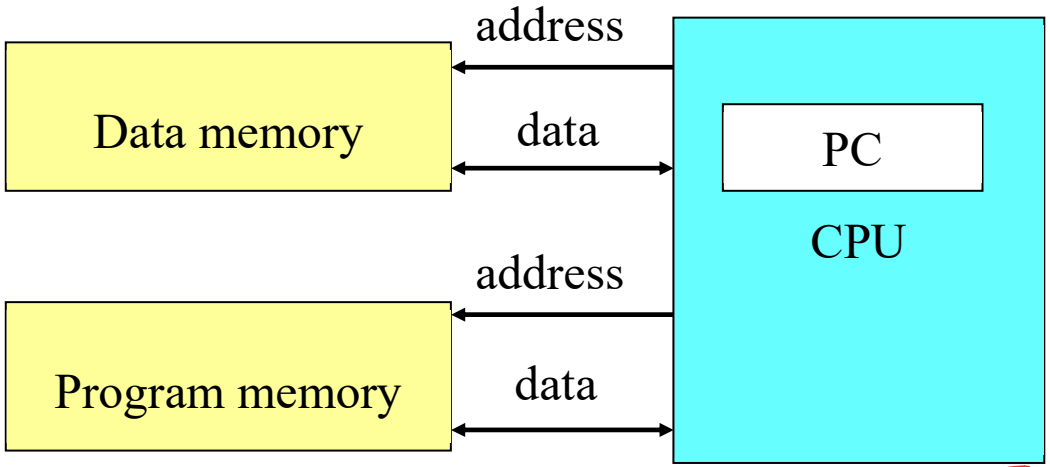
von Neumann architecture

- Memory holds data, instructions.
- Central processing unit (CPU) fetches instructions from memory.
- Separate CPU and memory distinguishes programmable computer.
- CPU registers help out: program counter (PC), instruction register (IR), general-purpose registers, etc.

- memory usually much slower than CPU
- bottleneck for system performance
- use small on CPU cache memory for much higher performance



Harvard Architecture



- Harvard architecture cannot use self-modifying code.
- It allows two simultaneous memory fetches.
- Most DSPs use Harvard architecture for streaming data.

self-mod code is useful for obfuscating code for competition reasons

- usually two caches (not L1, L2, but two seperate), but not neccessarily.

- possible to have different architecture for same instruction set. i.e. Intel vs AMD want same interface for third parties to use their products

Instruction Execution Process

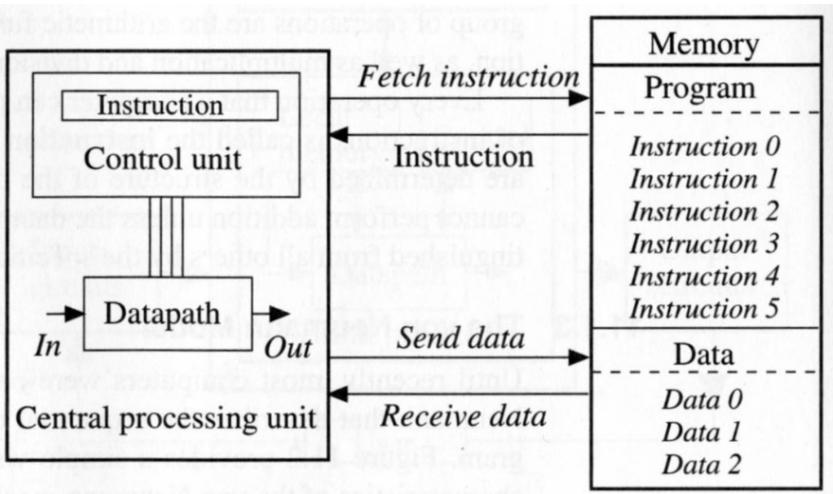
Instruction Fetch: Reads next instruction into the instruction register (IR). PC has the instruction address.

Instruction Interpretation: Decodes the op-code, gets the required operands and routes them to ALU.

Sequencing

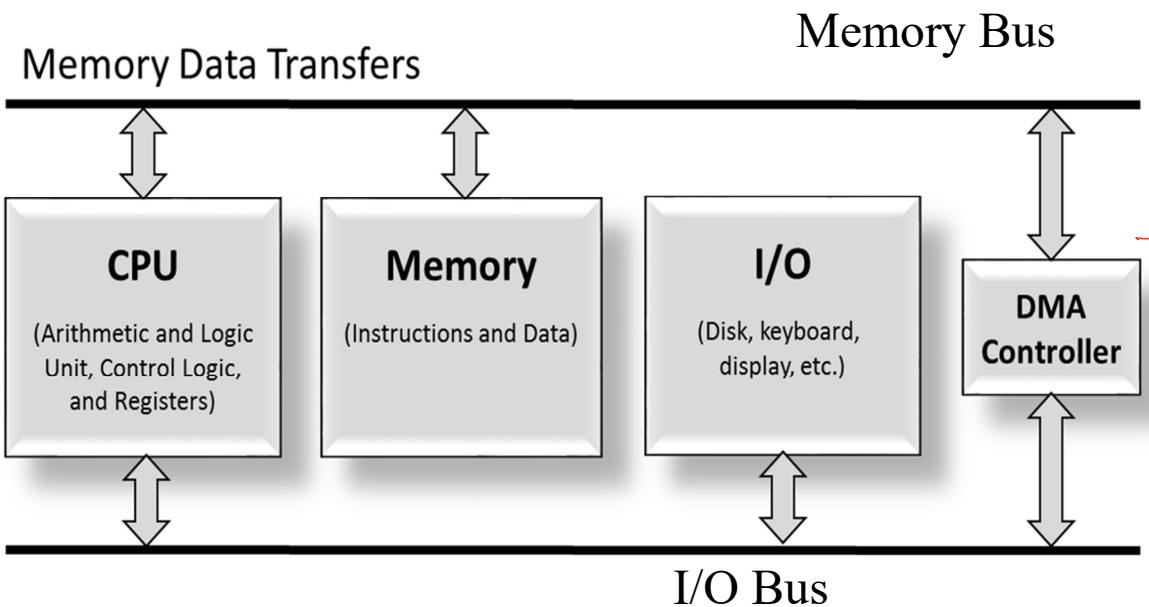
Determines the address of next instruction and loads it into the PC.

Execution: Generates control signals of ALU for execution.



System Organization

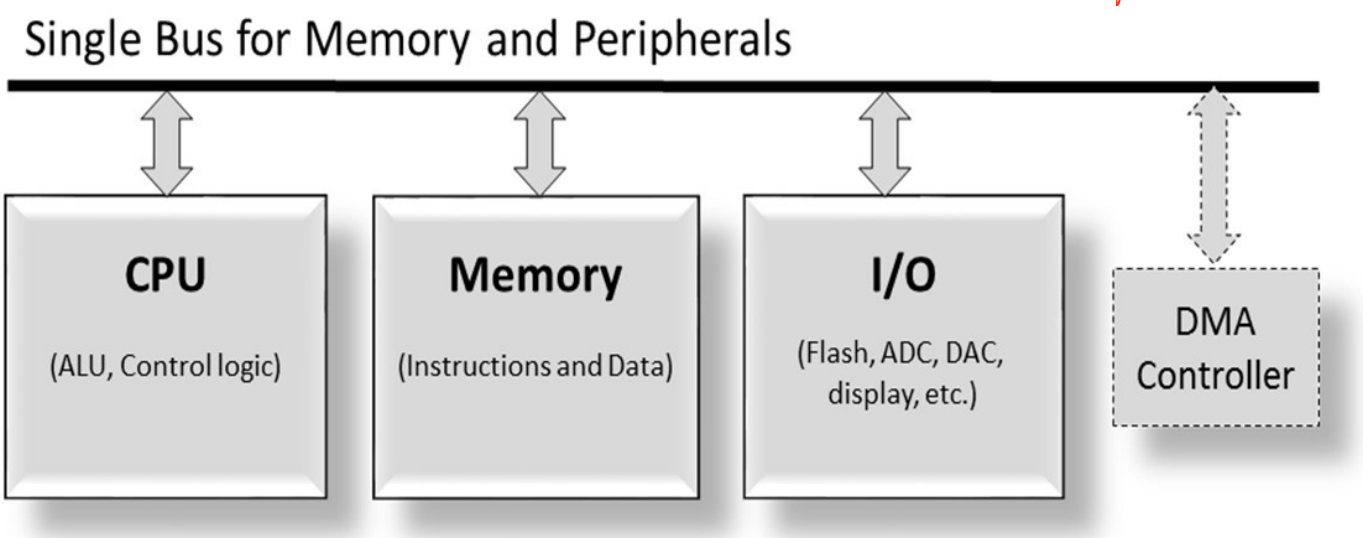
Memory and I/O having Separate Bus



– Controller which can help CPU transfer data between memory and I/O

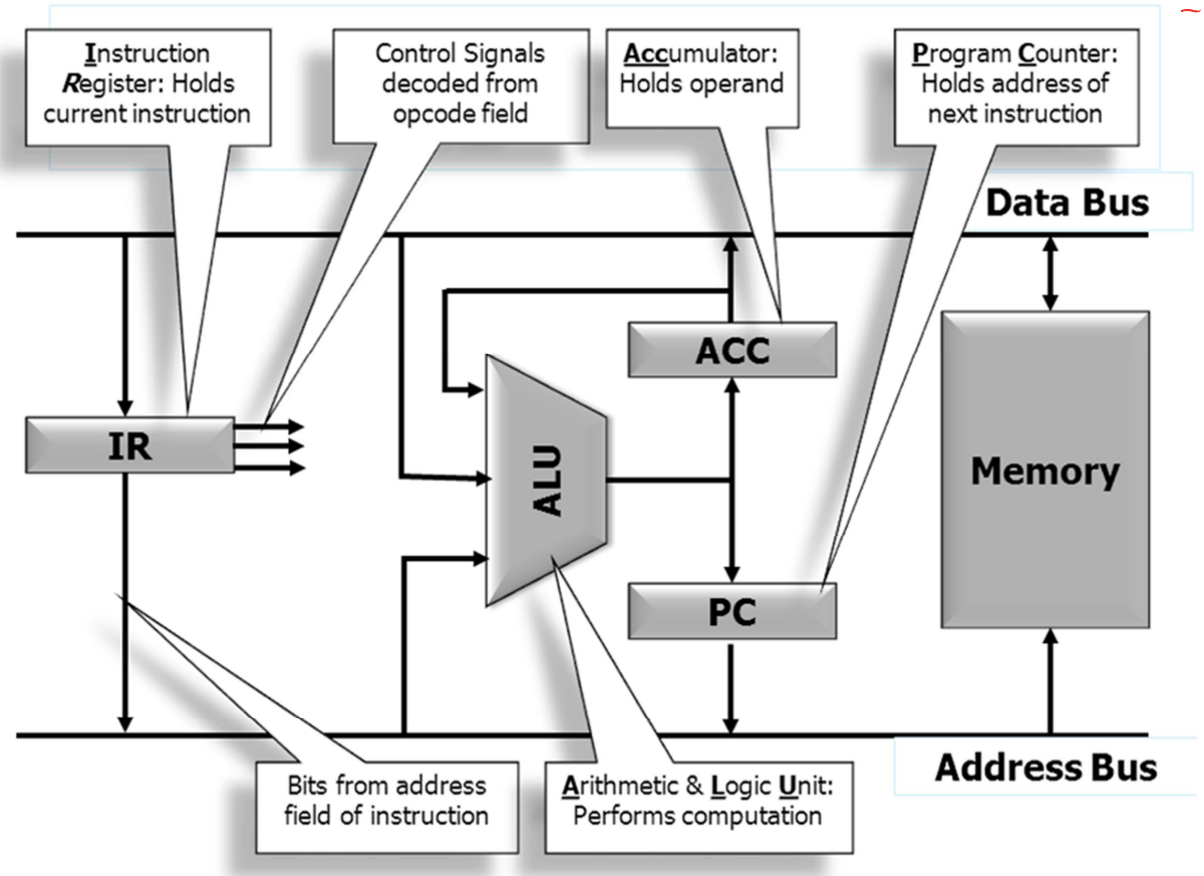
Memory Mapped Peripherals

- if reading/writing to specific memory address then peripherals are memory mapped
- make life easier for programming CPU.



Single Accumulator Architecture

- one of oldest architecture
- 90's CPUs



Interrupts

A computer program has only two ways to determine the conditions that exist in internal and external circuits.

- One method uses software instructions that jump to subroutine on some flag status.
- The second method responds to hardware signals called interrupts that force the program to call interrupt-handling subroutines.
- Interrupts take processor time only when action is required.
- Processor can respond to an external event much faster by using interrupts.

The whole programming of microcomputers and micro-controller by using interrupts is called real-time programming.

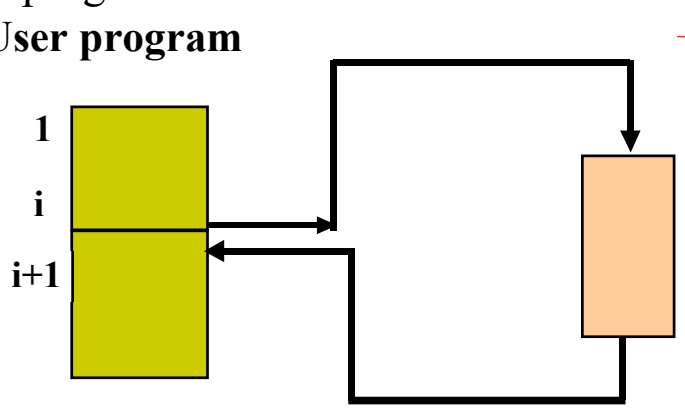
Interrupts are often the only way for successful real-time programming.

Instruction Cycle with Interrupts

Generally CPU checks for interrupts at the end of each instruction and executes the interrupt handler if required.

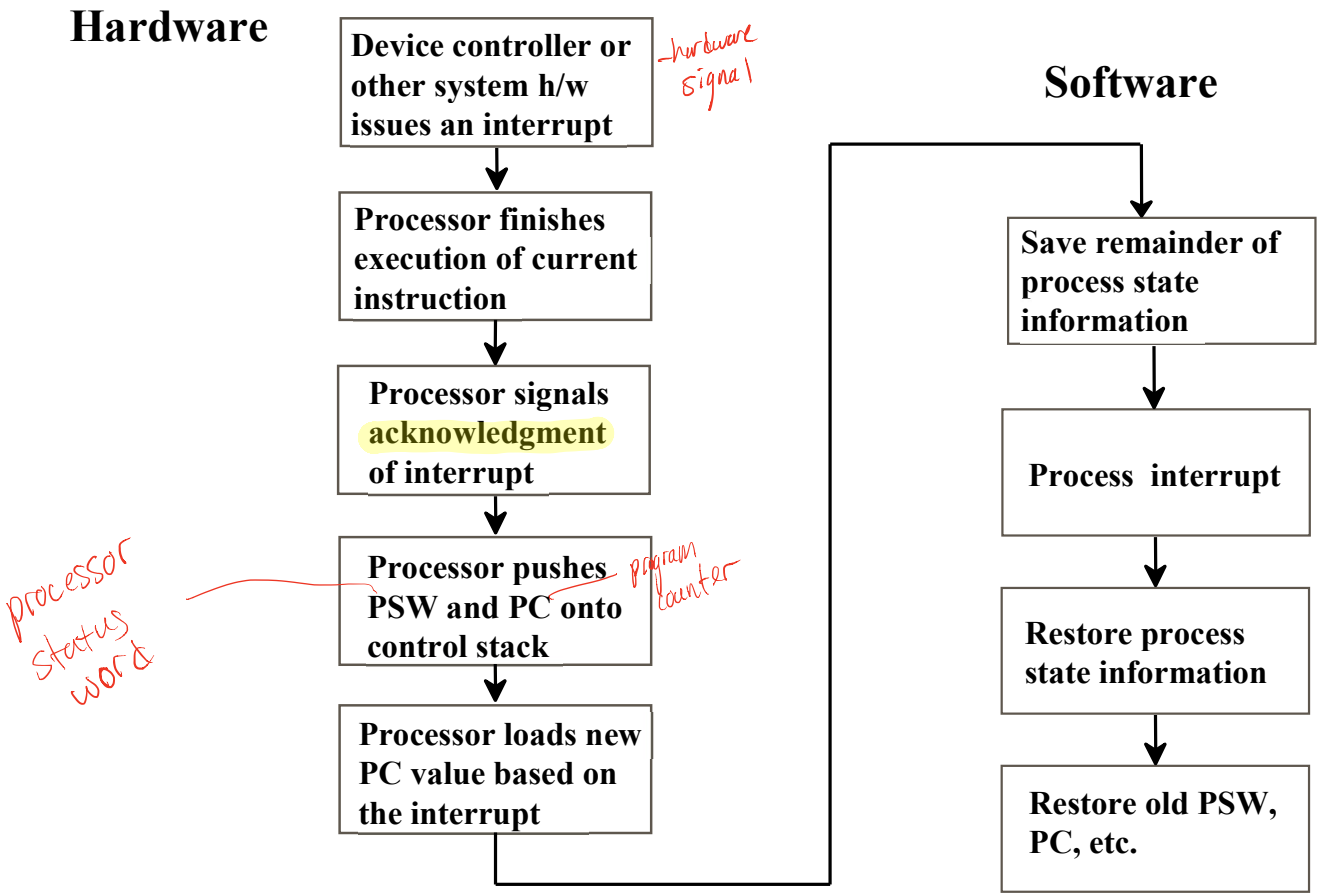
Interrupt Handler program identifies the nature/source of an interrupt and performs whatever actions are needed.

- It takes over the control after the interrupt.
- Control is transferred back to the interrupted program that will resume execution from the point of interruption.
- Point of interruption can occur anywhere in a program.
- State of the program is saved.



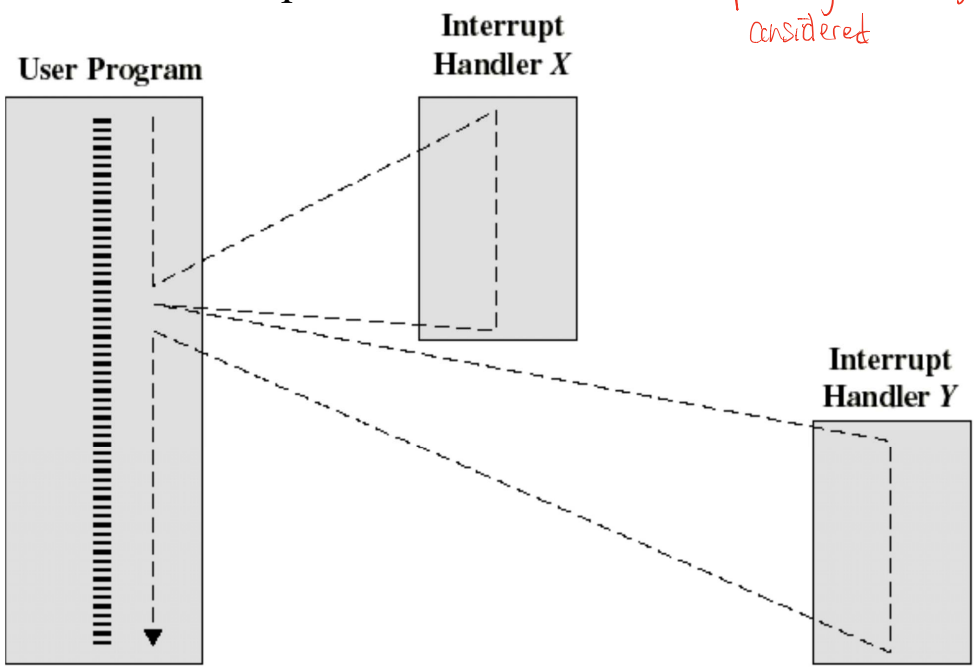
- the time it takes to service the interrupt should be very minimal (i.e. real-time)
(the time from interrupt coming in to handling)

Interrupt Processing



Multiple Interrupts (Sequential Order)

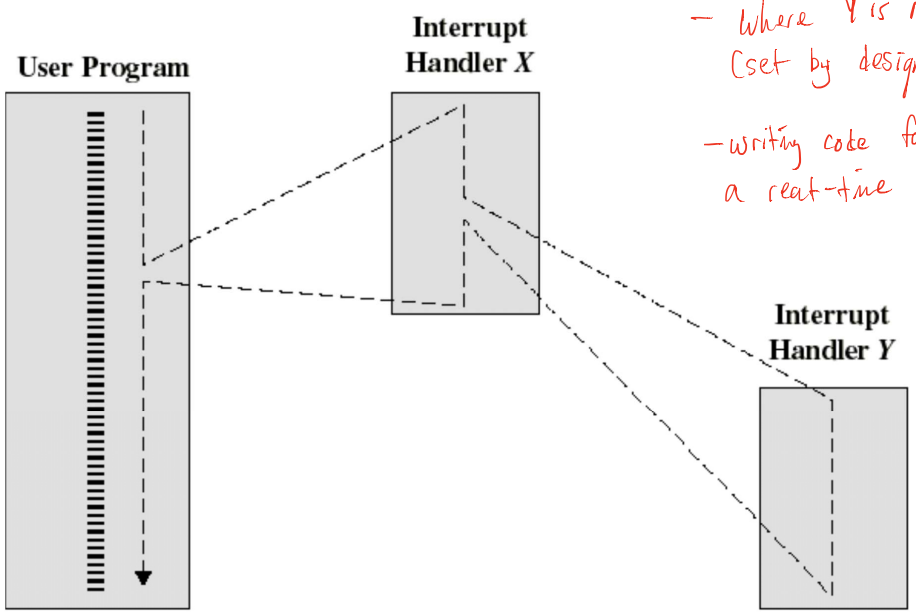
- Disable interrupts to complete the interrupting task at hand.
- Additional interrupts remain pending until interrupts are enabled. Then interrupts are considered in order
- After completing the interrupt handler routine, the processor checks for additional interrupts.



Multiple Interrupts (Nested)

- A higher priority interrupt causes lower-priority interrupts to wait.
- A lower-priority interrupt handler is interrupted.

For example, when input arrives from a communication line, it needs to be absorbed quickly to make room for additional inputs.



- where Y is more important than X (set by designer)

- writing code for embedded system without a real-time operating system is hard.

Processor Operation Modes

– a CPU must have user/supervisor mode to implement an operating system

User mode

- A user program is running.
- Certain instructions are not allowed.
- Memory mapping (base and bound) is enabled.

Supervisor mode

- The operating system is running.
- All instructions are allowed.
- Memory mapping (base and bound) is disabled.

A single PSW (processor status word) bit sets the above two modes:

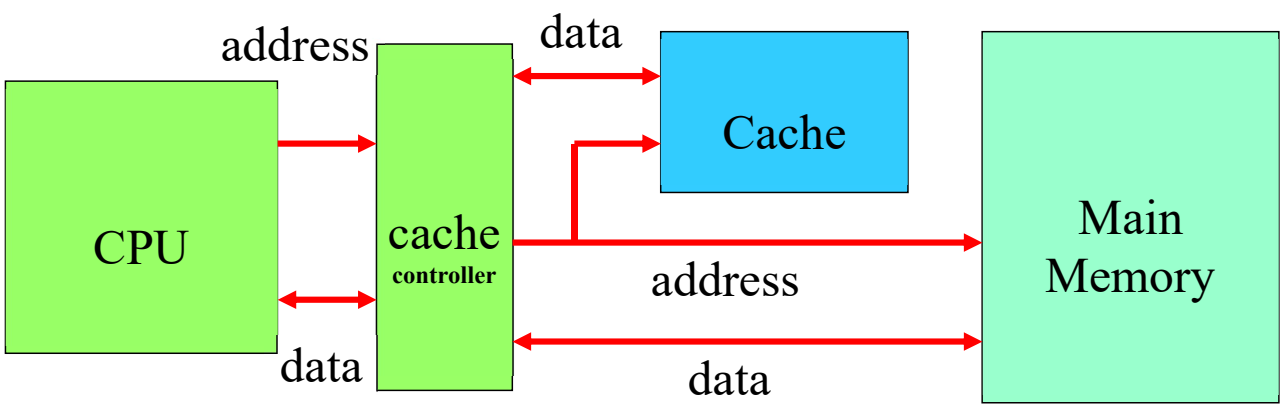
For instance: PSW-bit =1 for Supervisor mode

Cache Memory

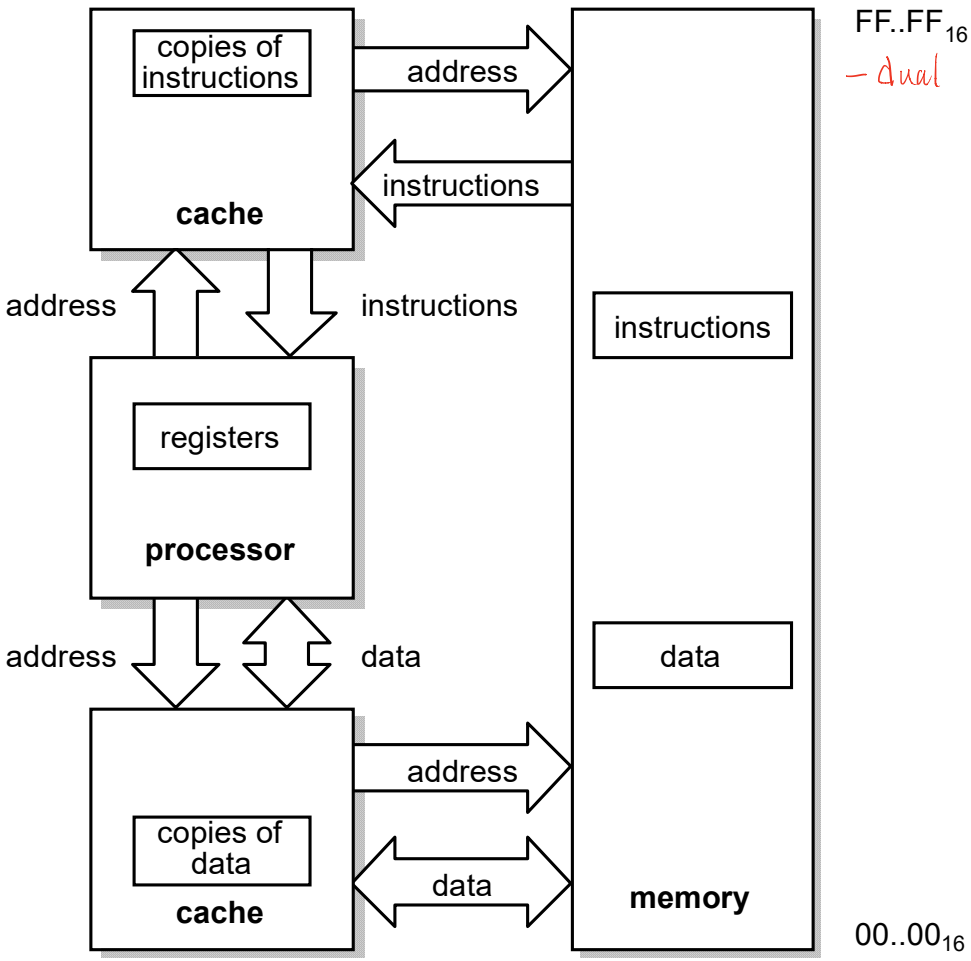
– theoretically embedded systems do not need a cache

- Cache: Expensive but very fast memory directly connected to CPU interacting with slower but much larger main memory.

- Processor first checks if the addresses word is in cache.
- If the word is not found in cache, a block of memory containing the word is moved to the cache.

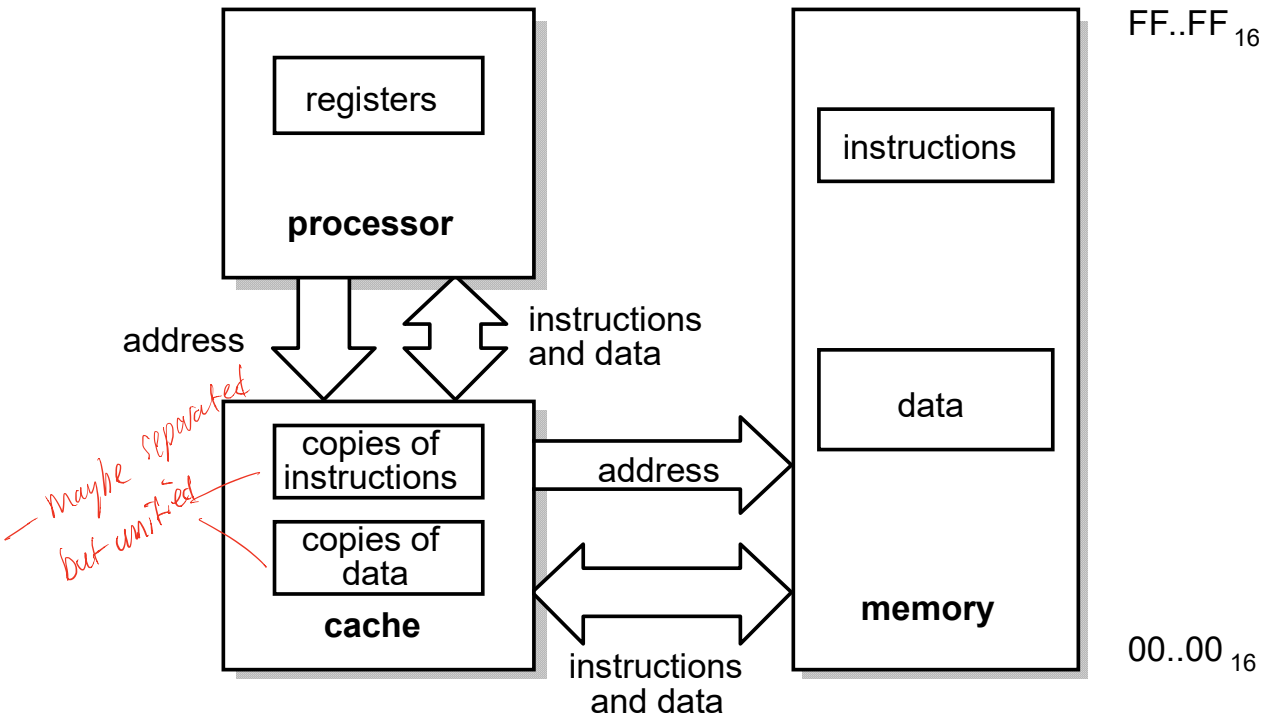


Separate Data and Instruction Caches (Harvard)



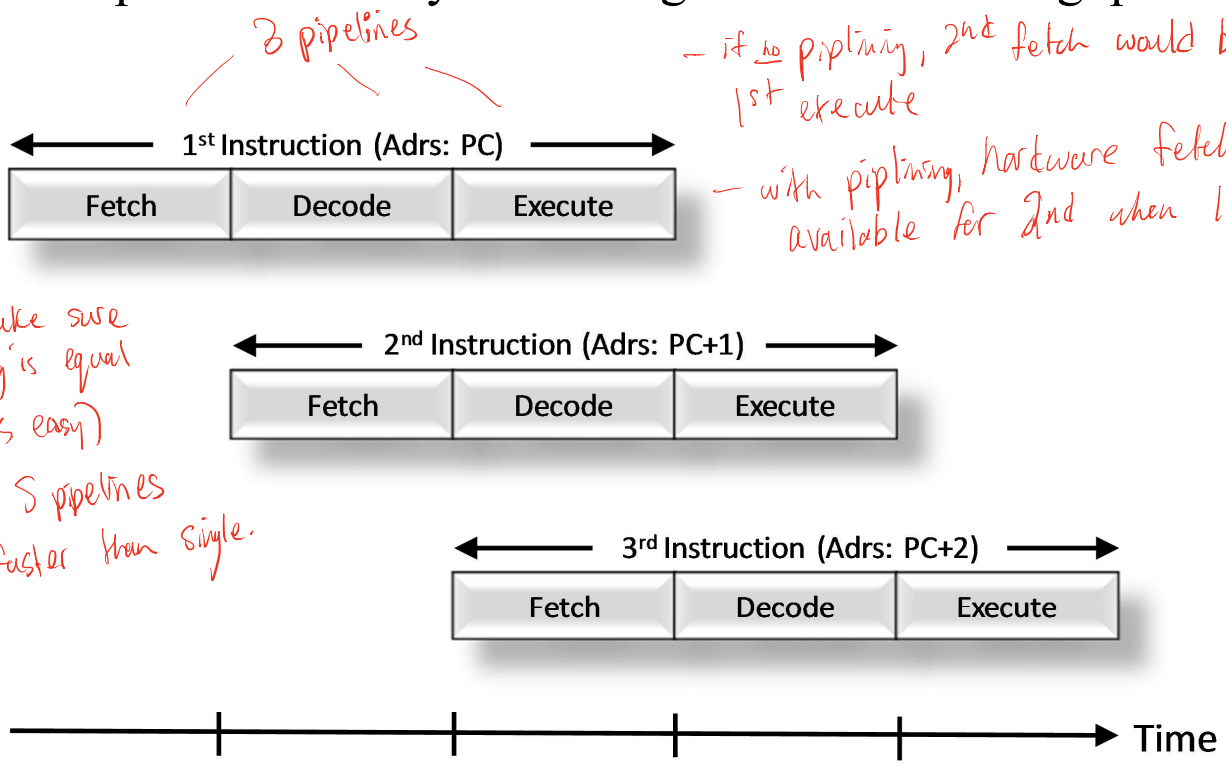
– dual port memory

A Unified Instruction and Data Cache



CPU Pipelining

Improve performance by increasing instruction throughput.



- if no pipelining, 2nd fetch would be after 1st execute
- with pipelining, hardware fetch is available for 2nd when 1st decodes
- have to make sure stage timing is equal (not always easy)
- theoretically 5 pipelines is 5x faster than single.

CPU Pipelining

What makes pipelining easy?

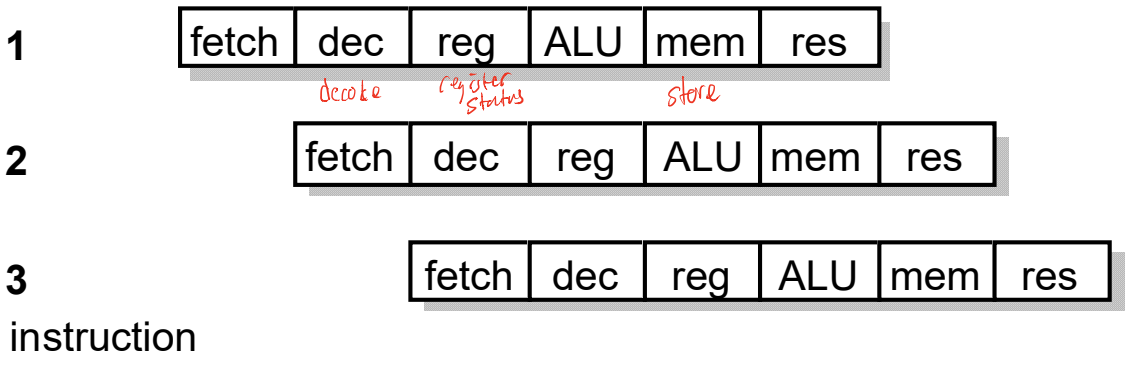
- When all instructions are of the same length. (RISC CPUs)
- Few instruction formats.
- Memory operands appear only in loads and stores.

What makes pipelining hard?

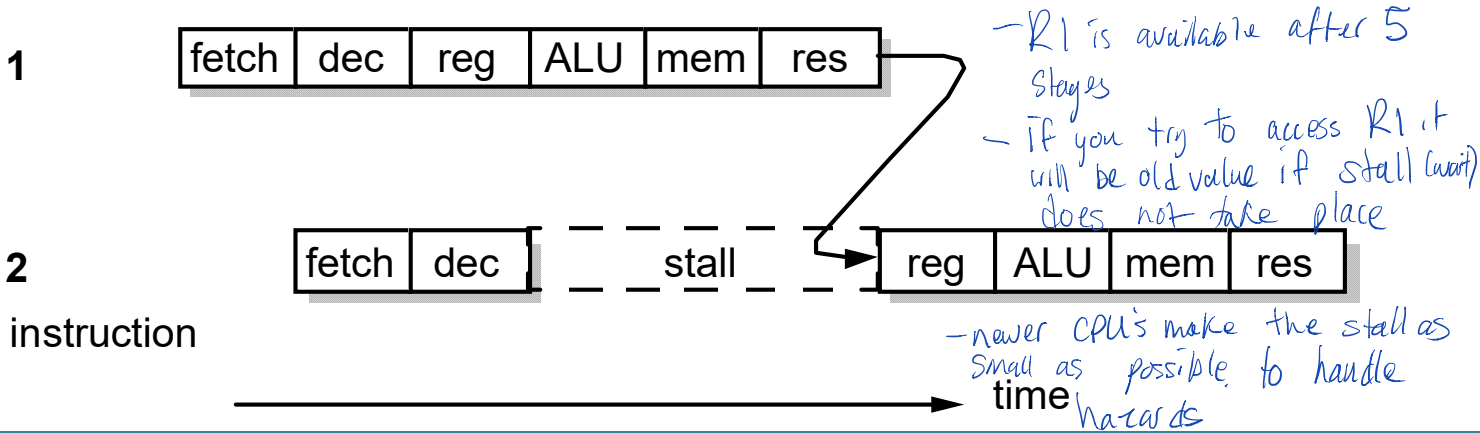
- **Structural Hazards:**
- **Control Hazards:**
- **Data Hazards**

CPU design course.

Pipelined Instruction Execution (5-stage Pipeline)



Read-after-write pipeline hazard



CPU Power Consumption

- Most modern CPUs are designed with power consumption in mind to some degree.
- Power vs. energy:
  - Heat depends on power consumption;
  - Battery life depends on energy consumption.

Power Saving Strategies

- Reduce power supply voltage.
- Run at lower clock frequency.

– Can also turn portions of CPU off, or disable power

– Memory read/writes consume power ⇒ minimize

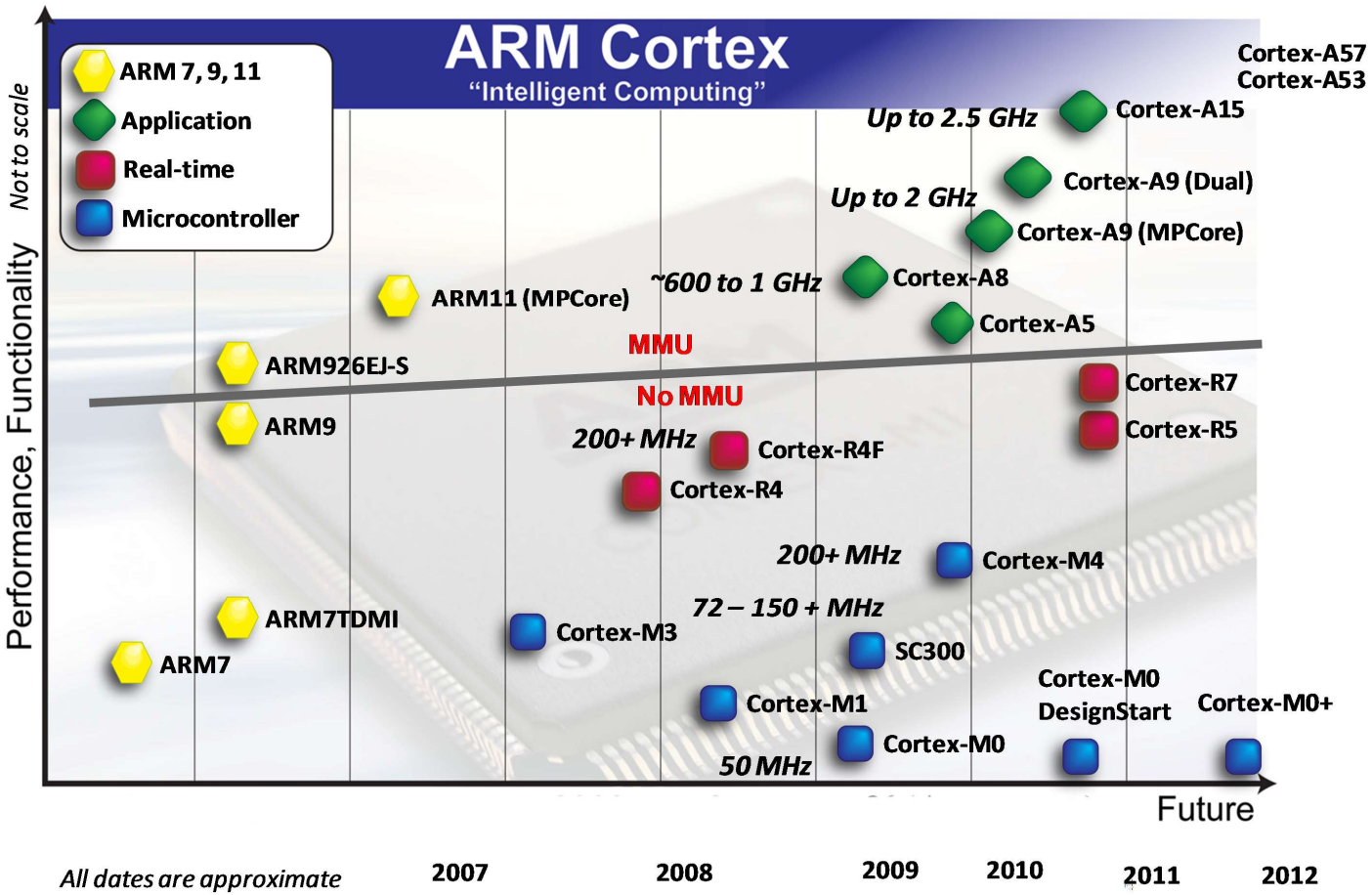
Microcontrollers and Embedded CPUs

Microcontrollers are available in 4 to 32-bit word sizes.
8-bit or 16-bit micro-controllers are widely used.

Popular Microcontrollers

Model	Pins I/O	RAM	ROM	Counter Timer	Remarks
Intel 8051	40:32	128	4K	2	128K External, Serial port
Motorola 6811	52:40	256	8K	2	A/D, Watch Dog timer, Serial Port
ARM CPUs TMS470-ARM	Soft	8-64KB	64KB-1MB		16/32-bit RISC DMA, Watch Dog timer, CAN, I^2C
Intel, 80C196	68:40	232	8K	2	16-bit 64K External, Serial, A/D, WD
Intel, 80960 (i960)	132	512 Ins Cache	--	20MHz Clock	32-bit bus, FPU, Interrupt control No I/O on-chip
MIPS32 4K	Soft core	**I/D Cache**	--		32-bit RISC, EJTAG and On-chip bus
MC68360 QUICC	240/241			4 or 16-bit	32-bit, WD, Fault-tolerant, 4-Ethernet, 2 serial, Integrated Com Controller
Nios II	Soft core	**On-chip Ram**		IP	32-bit RISC, Soft Core for Embedded CPUs on FPGA

ARM Powered Products


ARM Cortex
"Intelligent Computing"
ARM 7, 9, 11
Application
Real-time
Microcontroller
Not to scale
Performance, Functionality
Cortex-A57
Cortex-A53
Up to 2.5 GHz
Cortex-A15
Cortex-A9 (Dual)
Up to 2 GHz
Cortex-A9 (MPCore)
~600 to 1 GHz
Cortex-A8
ARM11 (MPCore)
Cortex-A5
MMU
ARM926EJ-S
No MMU
Cortex-R7
ARM9
Cortex-R5
200+ MHz
Cortex-R4F
Cortex-R4
200+ MHz
Cortex-M4
72 – 150 + MHz
ARM7TDMI
Cortex-M3
SC300
ARM7
Cortex-M0
DesignStart
Cortex-M0+
Cortex-M1
Cortex-M0
50 MHz
Future
All dates are approximate
2007
2008
2009
2010
2011
2012

ARM CPU

Versions, cores and architectures ?

- What is the difference between ARM7™ and ARMv7 ?
- ARM doesn't make chips….well maybe a few test chips.

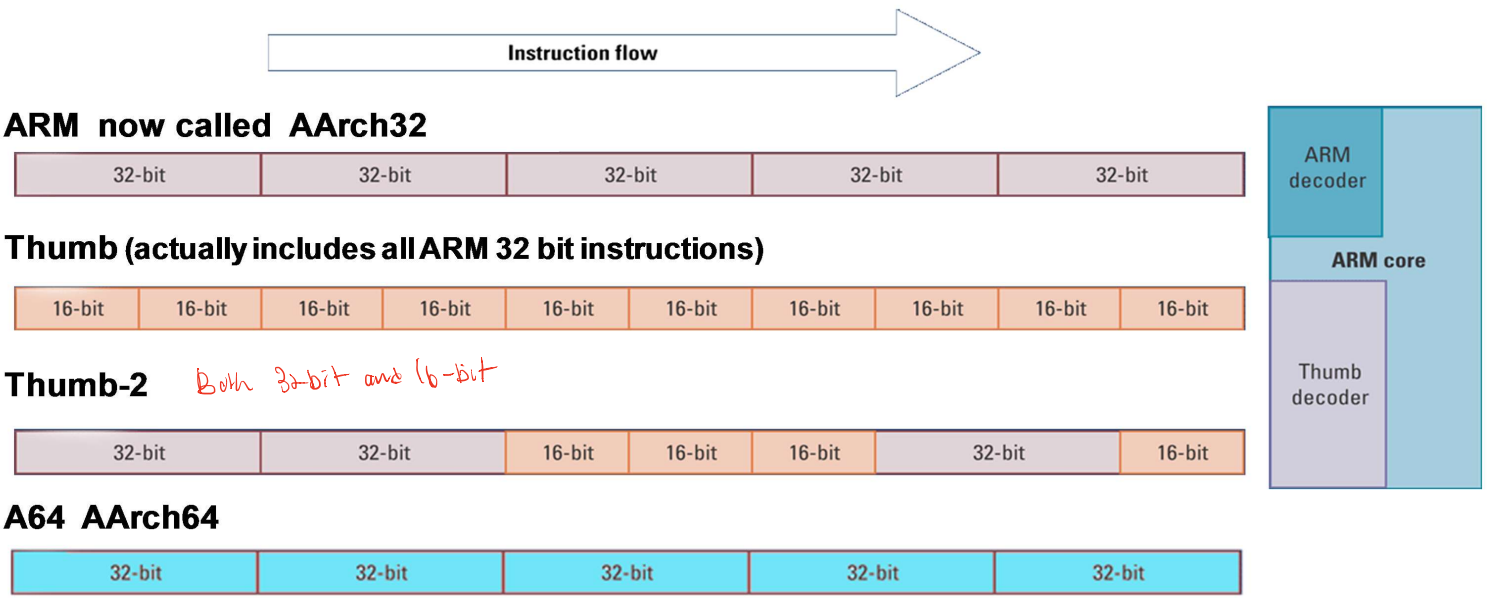
Family	Architecture	Cores
ARM7TDMI	ARMv4T	ARM7TDMI(S)
ARM9 ARM9E	ARMv5TE(J)	ARM926EJ-S, ARM966E-S
ARM11	ARMv6 (T2)	ARM1136(F), 1156T2(F)-S, 1176JZ(F), ARM11 MPCore™
Cortex-A	ARMv7-A	Cortex-A5, A7, A8, A9, A15
Cortex-R	ARMv7-R	Cortex-R4(F)
Cortex-M	ARMv7-M	Cortex-M3, M4
	ARMv6-M	Cortex-M1, M0
NEW !	ARMv8-A	64 Bit

ARM Processor Licenses (the public ones)

- ARMv8-A ? NVIDIA, Applied Micro, Cavium, AMD, Broadcom, Calxeda, HiSilicon, Samsung and STMicroelectronics
- Cortex-A15 4 ST-Ericson, TI, Samsung, nVIDIA
- Cortex-A9 9 NEC, nVIDIA, STMicroelectronics, TI, Toshiba …
- Cortex-A8 9 Broadcom, Freescale, Matsushita, Samsung, STMicroelectronics, Texas Instruments, PMC-Sierra
- Cortex-A5 3 AMD,
- Cortex-R4(F) 14 Broadcom, Texas Instruments, Toshiba, Inf
- Cortex-M4 5 Freescale, NXP, Atmel, ST
- Cortex-M3 29 Actel, Broadcom, Energy Micro, Luminary Micro, NXP, STMicroelectronics, TI, Toshiba, Zilog, …
- Cortex-M0 14 Austria-microsystems, Chungbuk Technopark, NXP, Triad Semiconductor, Melfas
- Cortex-M0+ Freescale, NXP
- ARM7 172, ARM9 271, ARM11 82

ARM Instruction Sets

- ARM (32 bit) now referred as AArch32
- Thumb (16 bit)
- Thumb2: Cortex-Mx processors. Cortex-R, A have Thumb2 + ARM.
- A64 (64 bit) referred as AArch64

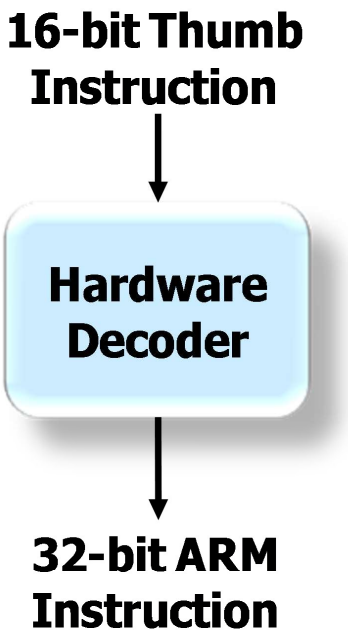


Tools: Keil MDK™ with µVision:

- For Cortex-M and Cortex-R processors.
- Proprietary IDE µVision
- ARM compiler, assembler and linker.
- ULINK2, ULINK*pro*, CMSIS-DAP + more debug adapters.
- Many board support packages (BSP) and examples.
- MDK Professional: TCP/IP. CAN, USB & Flash middleware.
- Serial Wire Viewer and ETM, MTB & ETB Trace supported.
- Evaluation version is free from www.keil.com/arm.
- Is complete turn-key package: no add-ons needed to buy.
- Valuable technical support included for one year. Can be easily extended.
- Keil RTX RTOS included free with source code.

Three Types of Instruction Sets

- ARM Instruction Set
  - Instructions are 32 bits wide
  - Original RISC (lots of parallelism)
  - "Load/Store" Architecture
- Thumb Instruction Set — for smaller / less complicated devices
  - Subset of ARM instructions, some restrictions
  - Instructions are 16 bits wide (more like CISC)
  - Intended for compilers
  - Less parallelism, longer instruction sequences
  - but total code size is 30% smaller
- Jazelle Instruction Set
  - Java byte codes



The Wall Street Journal July 18, 2016

SoftBank to Buy ARM Holdings for $32 Billion

Japanese group's deal for U.K.-based chip designer comes together in just two weeks

Three Instruction Sets (RISC)

	ARM	Thumb*	Jazelle
Instruction Size	32 bits	16 bits	8 bits
Core instructions	58	30	> 60% of Java byte codes in hardware; rest in software
Conditional Execution	most	Only branch instructions or in an IT block	N/A
Data processing instructions	Access to barrel shifter and ALU	Separate barrel shifter and ALU instructions	N/A
Program status register	Read/write in privileged mode	No direct access	N/A
Register usage	15 general purpose registers + pc	8 general purpose registers + 7 high registers + pc	N/A

ARMv7 vs Thumb Instruction Set

- ARM has 2 instructions sets = traditional ARM ISA (i.e. ARMv7, etc) and the condensed Thumb ISA (16/32bit)
- ARMv7 is a 32-bit standard instruction set derived by ARM
- **Thumb2** incorporates 16/32-bit mix by using a Unified Assembler Language (UAL) to alternate between theese 2 ISAs, basically adds suffixes the instructions ['N' (16) and 'W' (32)] to let the compiler know if it's 16-bit or 32-bit
- Each Thumb instruction directly correlates to an equivalent 32-bit ARM instruction version, which has the same effect on the processor and only their encodings are different.

16 bit **Thumb** | 0| 0 | 0| OP (2b) | Imm5 | Rs (3b) | Rd (3b)

32 bit **ARM** | cond 4b | 0 | 0| 1| op (4b) | S | Rn (4b) | Rd (4b) | rotate (4b) | imm8

Thumb when in 'N' mode has less register space to work with, and can only deal with smaller imm values (for loads/stores, branching etc.)

HISTORY of ARM ISA

ARM initially came out with Thumb, and then introduced Thumb-2 technology (still using Thumb ISA) bc ARM had two separate syntaxes which made compilation and execution more complex.

- Solved with UAL
- Using Thumb only one ISA could be active at a time, switching in the middle of an application gives some overhead (i.e. BLX)
- Thumb is used to trim down code size and memory utilization with hardly no change to the core

Decreased code size = less memory usage = lower power consumption

Reduces code size by > 30%

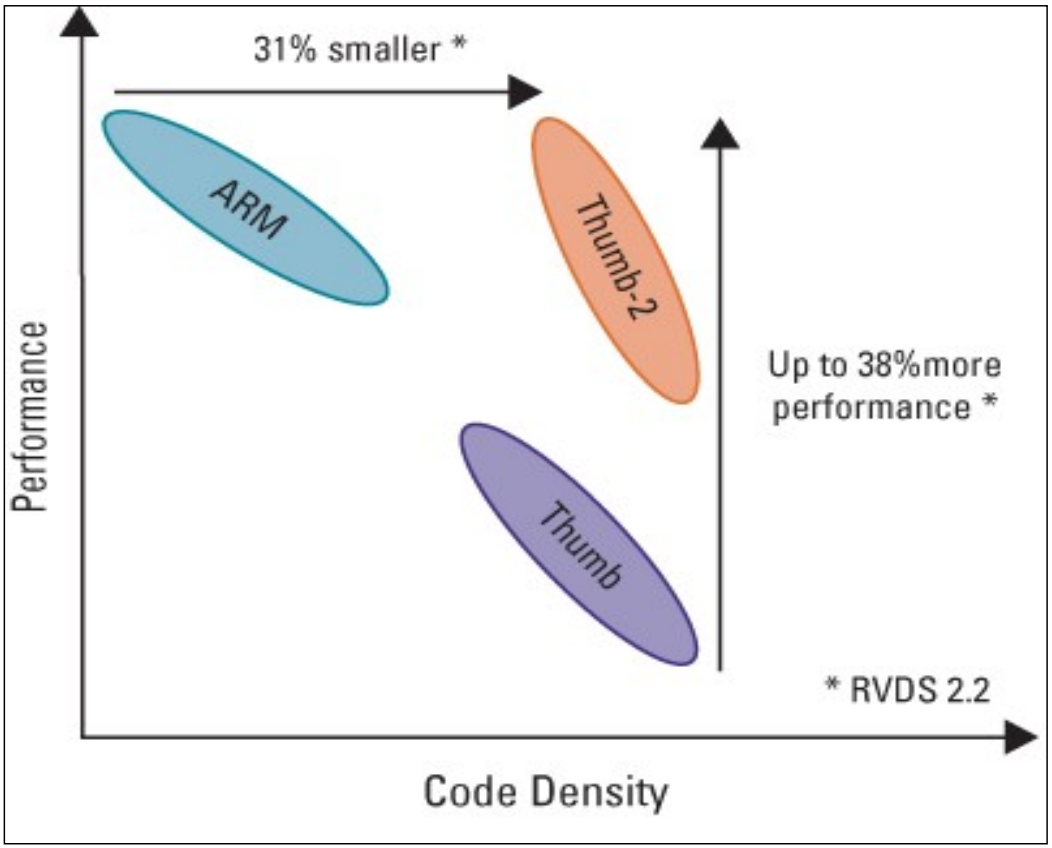
Thumb still has 32-bit bus and operates with a 32bit data-width.

More instructions can be fetched from memory as instructions are smaller and code size is condensed. In general, fetching instructions is slower than executing instructions

* Less instruction memory utilization

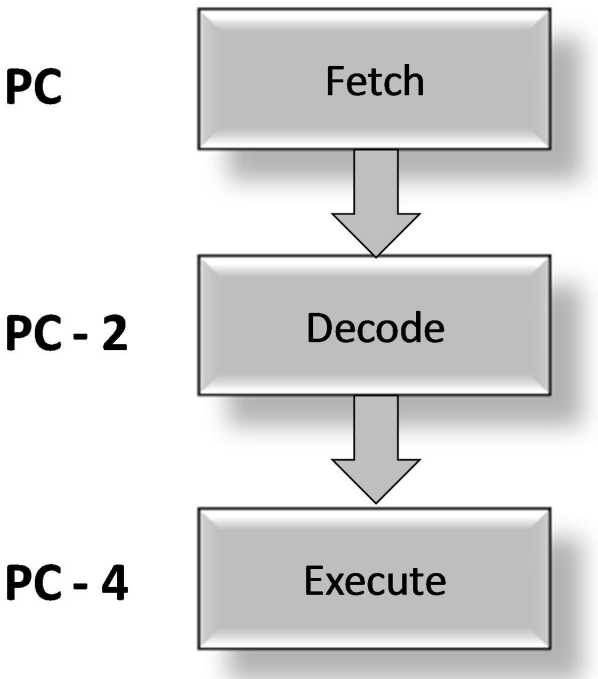
* Very little performance difference.

Performance/Code for Instruction Sets



-Use Thumb for limited memory as well as low power.

Pipelined Instruction Fetch, Decode and Execute



16-bit Instruction fetched from memory

Decompress thumb instruction, Decode ARM instruction
Select registers

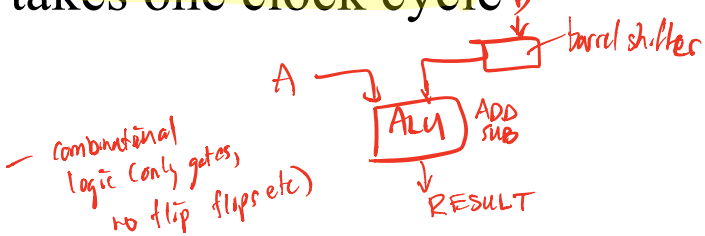
Read register(s) from Register Bank,
Shift and ALU operation,
Write register(s) back to Register Bank

ARM7 Architecture

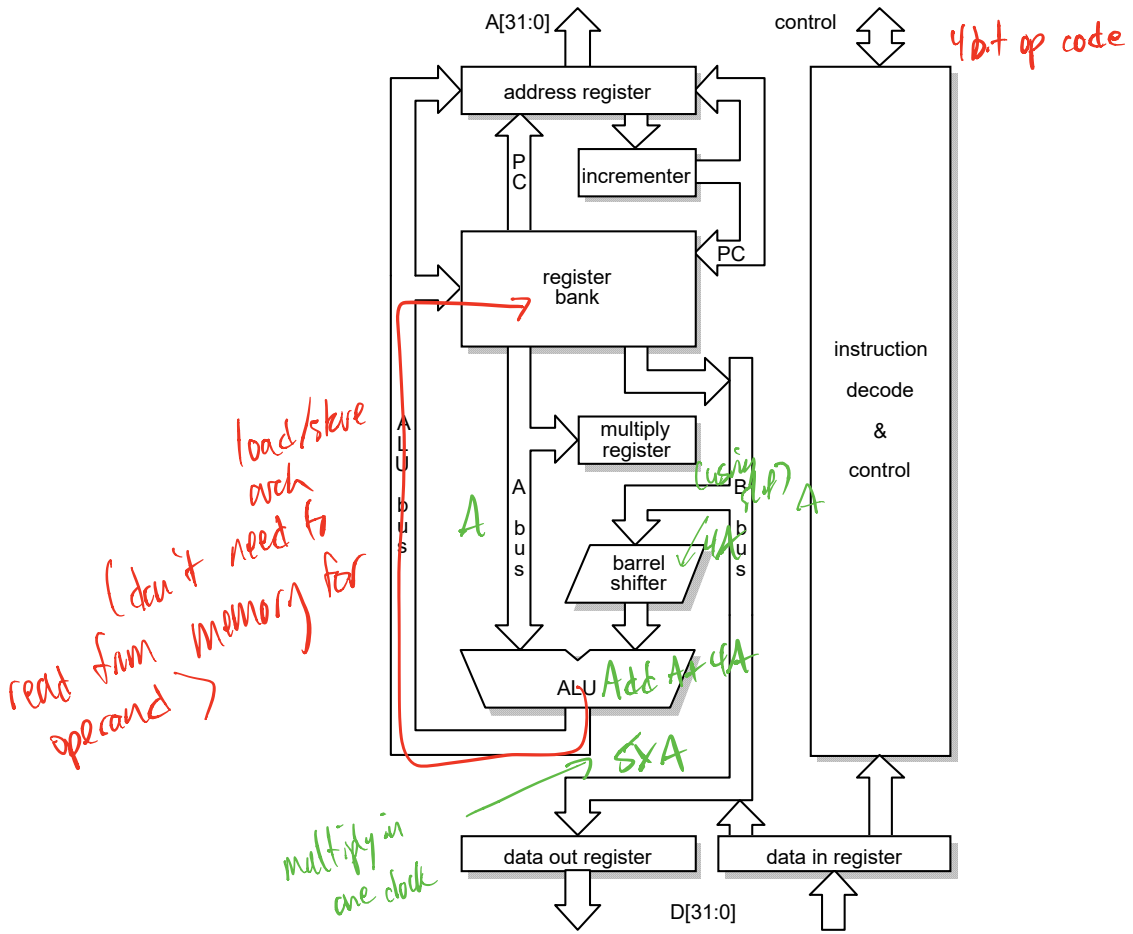
- Load/store architecture
- Most instructions are RISCy
  Some multi-register operations take multiple cycles
- All instructions can be executed conditionally

ARM7 is a small, low power, 32-bit microprocessor.
Three-stage pipeline, each stage takes one clock cycle

- Instruction fetch from memory
- Instruction decode
- Instruction execution.
  - Register read
  - A shift applied to one operand and the ALU operation
  - Register write



ARM CPU Core Organization



ARM7 Features (Also true for ARM-M3)

Combined Shift and ALU Execution Stage

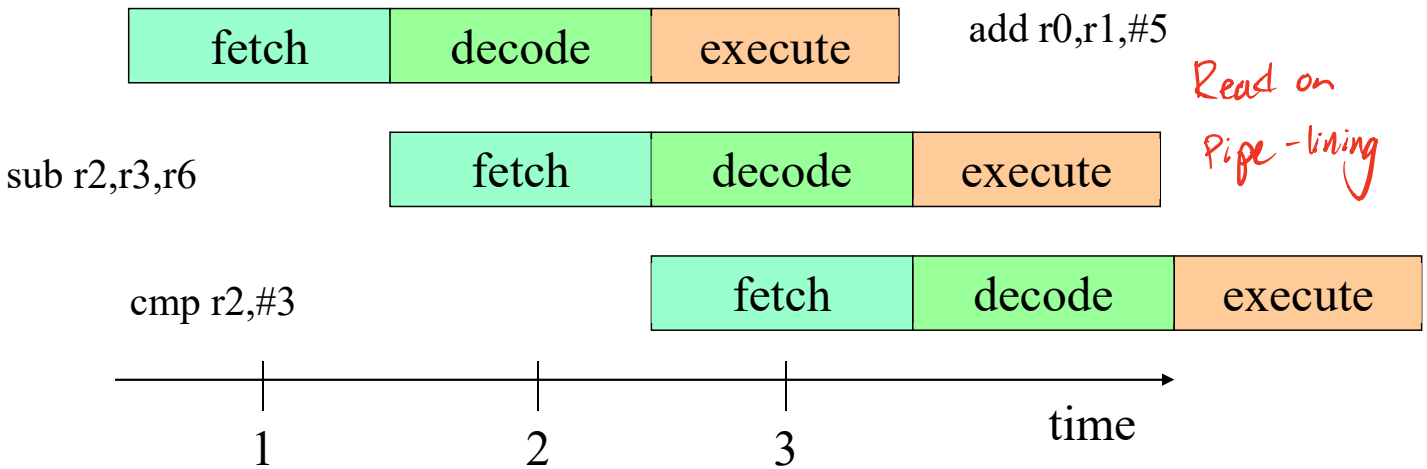
- A single instruction can specify one of its two source operands for shifting or rotation before it is passed to the ALU (shift + add one clock)
- Allows very efficient bit manipulation and scaling code
- Eliminates virtually single shift instructions from ARM code. ARM7 CPU does not have explicit shift instructions.
- A move instruction can apply a shift to its operand

ARM7 uses von-Neumann memory (one memory) architecture where instructions and data occupy single address space that can limit the performance

- Instruction fetching (and execution) must stop for instructions that access memory
- The pipeline stalls during load and store operations, ARM7 can continue useful work.

M3 has Harvard architecture.

ARM7 Pipeline Execution



- **Latency**
  Time it takes for an instruction to get through the pipeline.
- **Throughput**
  Number of instructions executed per time period.

ARM CPU Features: Modified RISC

Multiple Load and Store Operation

Reduce the penalty of data accesses during a stall in the pipeline
Multiple load/store instructions can move any of the ARM registers to and from memory, and update the memory address register automatically after the transfer.

- This not only allows one instruction to transfer many words of data (in a single bus burst), it also reduces the amount of instructions needed to transfer data.
- Make the ARM code smaller than other 32-bit CPUs
- These instructions can specify an update of the base address register with a new address after the transfer.

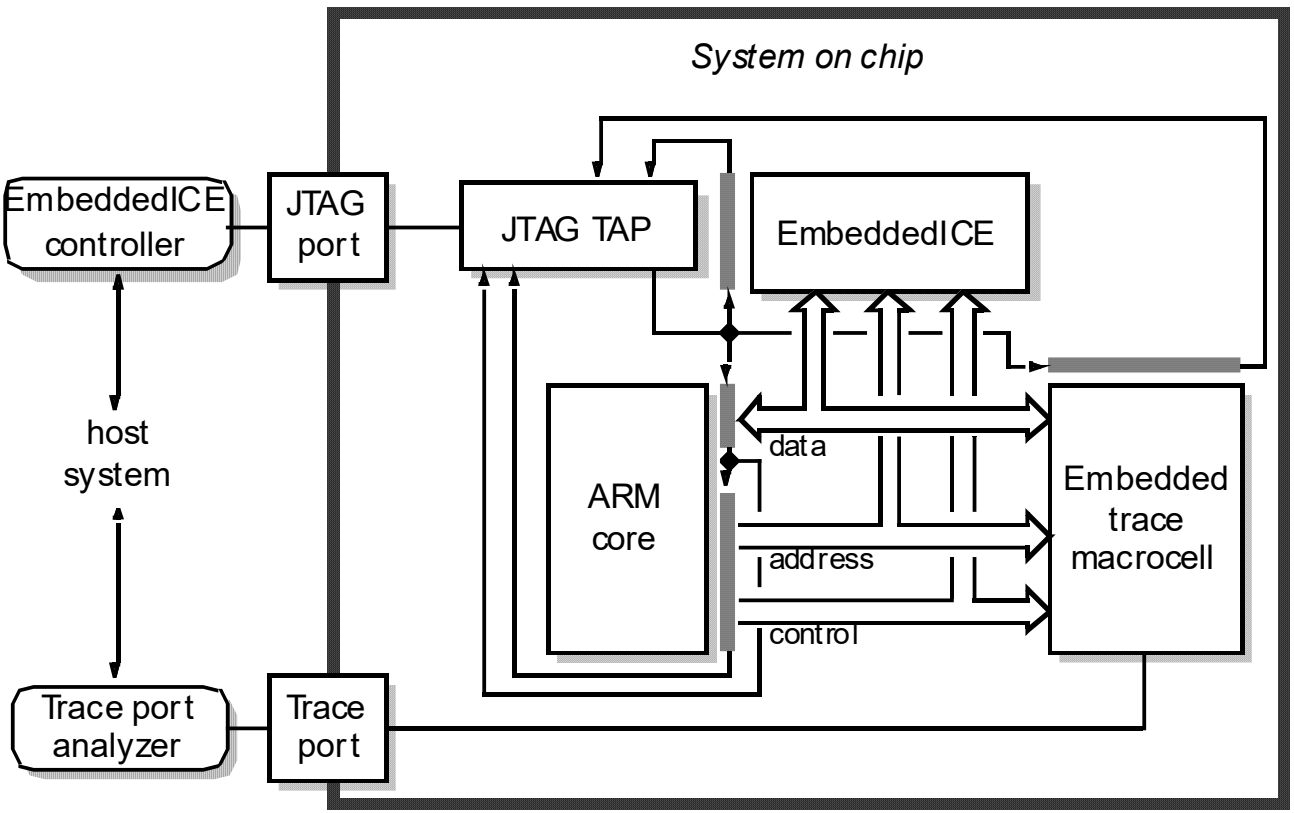
RISC CPU architectures would normally use a second instruction (add or subtract) to form the next address in a sequence.
ARM does it automatically with a single bit in the instruction, again a useful saving in code size.

ARM CPU (More) Features

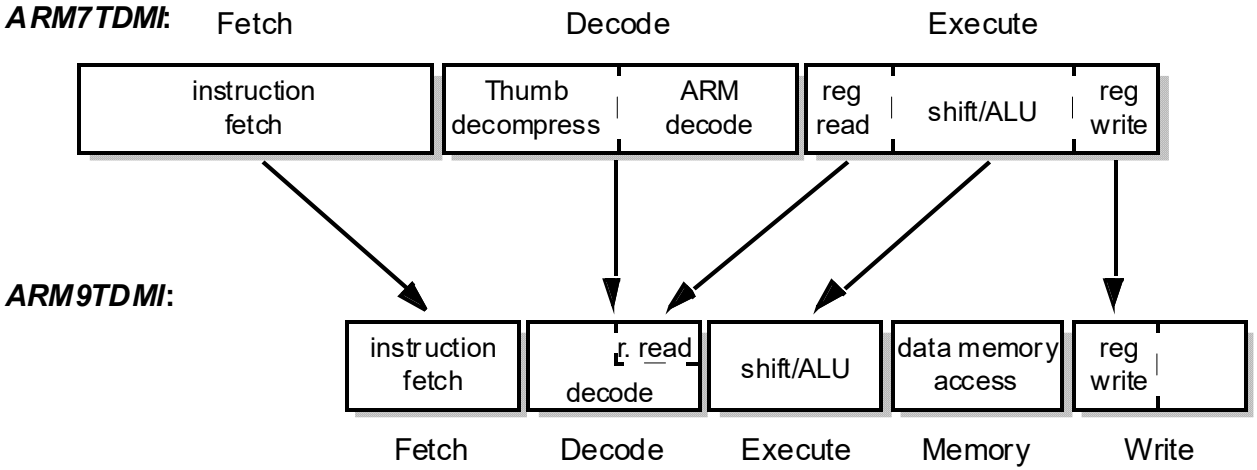
All instructions are conditionally executed:

- A very useful feature
- Loads, stores, procedure calls and returns, and all other operations can execute conditionally after some prior instruction to set the condition code flags
- Any ALU instruction may set the flags
- This eliminates short forward branches in ARM code
- It also improves code density and avoids flushing the pipeline for branches and increase execution performance
  - Most CPU architectures have conditional branch instructions
  - These follow a test or compare instruction to control the flow of execution through the program
  - Some architectures also have a conditional move instruction, allowing data to be conditionally transferred between registers

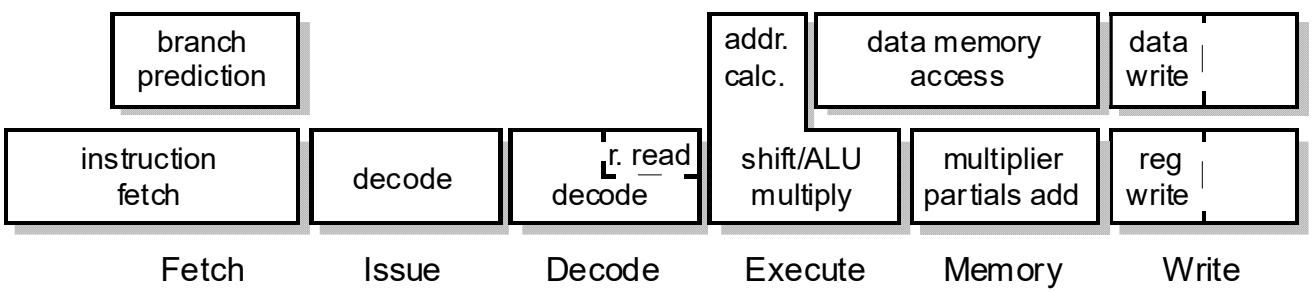
Real-timed Debug System Organization
(ARM7TDMI)



ARM7TDMI and ARM9TDMI Pipeline



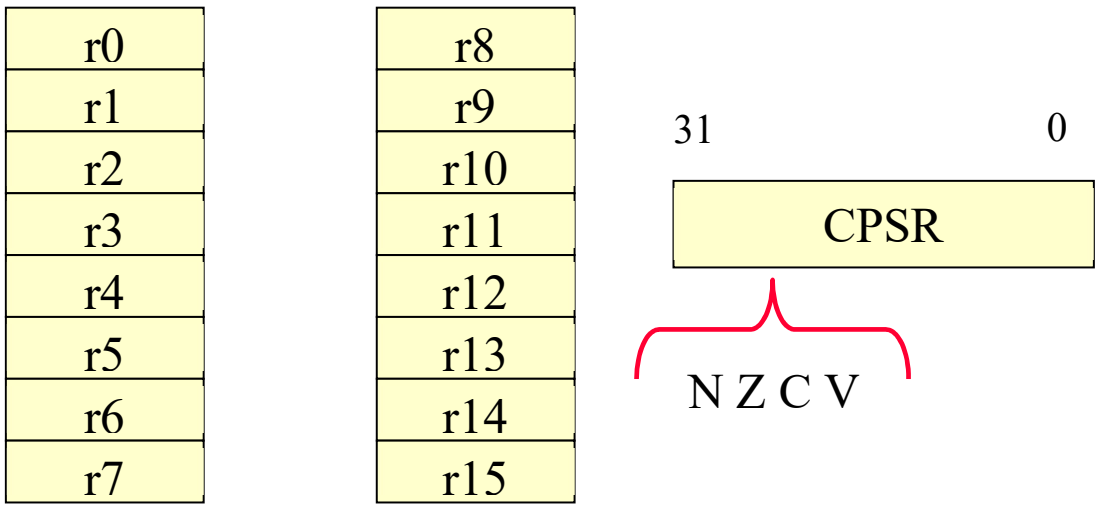
The ARM10TDMI pipeline



ARM Architectures

Core	Architecture
Classic ARM Processors	
ARM1	v1
ARM2, ARM2as, ARM3	v2, v2a
ARM6, ARM600, ARM610	v3
ARM7TDMI, ARM710T, ARM720T, ARM740T	v4T
ARM8, ARM810	v4
ARM9TDMI, ARM920T, ARM940T	v4T
ARM9ES	v5TE
ARM10TDMI, ARM1020E	v5TE
ARM11	v6
. . . . .	
ARM Cortex Processors	
ARM Cortex-M3	v7M
ARM Cortex-M4	v7ME
ARM Cortex-R4, R5, R7	v7R
ARM Cortex-A5, A8, A9, A15	v7A

ARM7: Programming Model

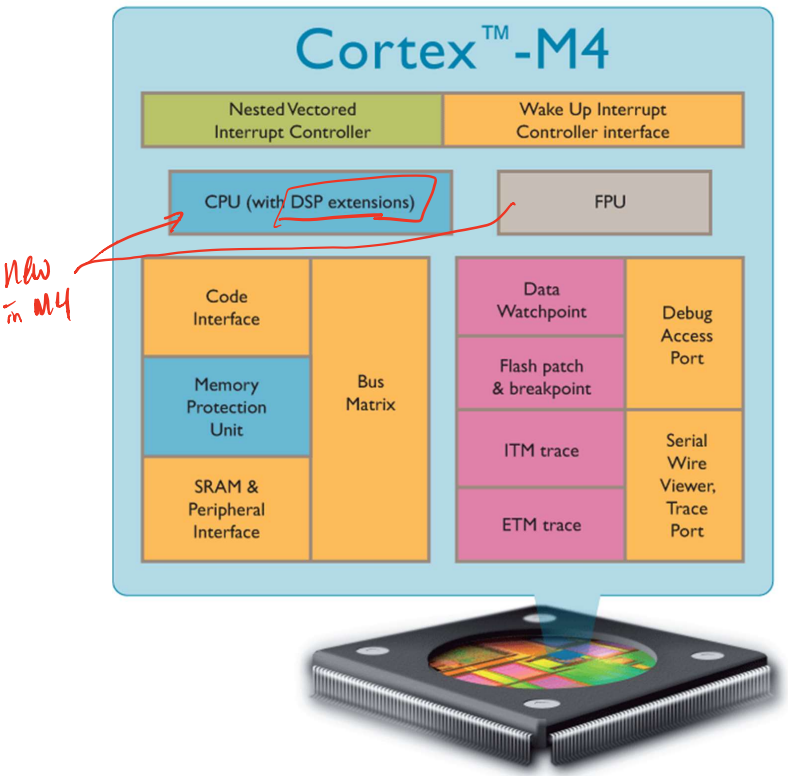


- Word is 32 bits long.
- Word can be divided into four 8-bit bytes.
- ARM addresses can be 32 bits long.
- Address refers to byte.
  Address 4 starts at byte 4.

PC +4 gets incremented by bytes.

ARM Cortex-M4

Latest Cortex-M series CPU that has a combination of efficient signal processing and low-power.

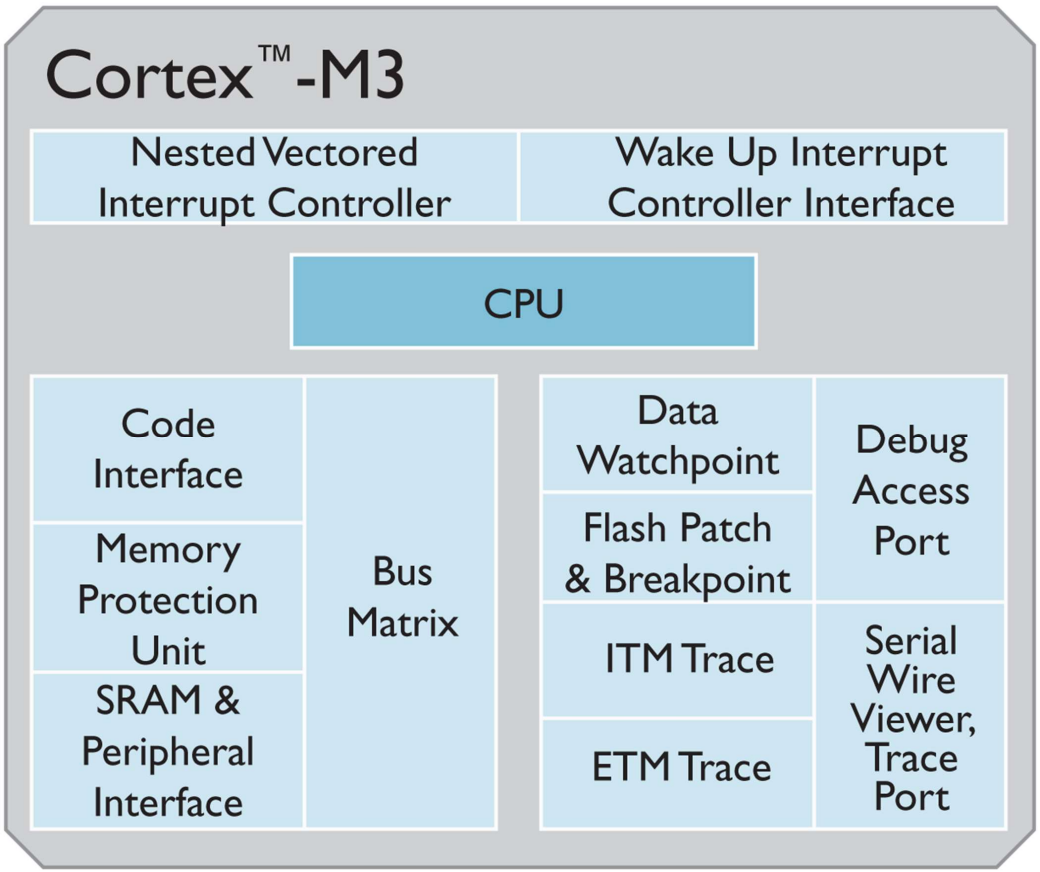


Harvard vs. Van Neumann Architecture

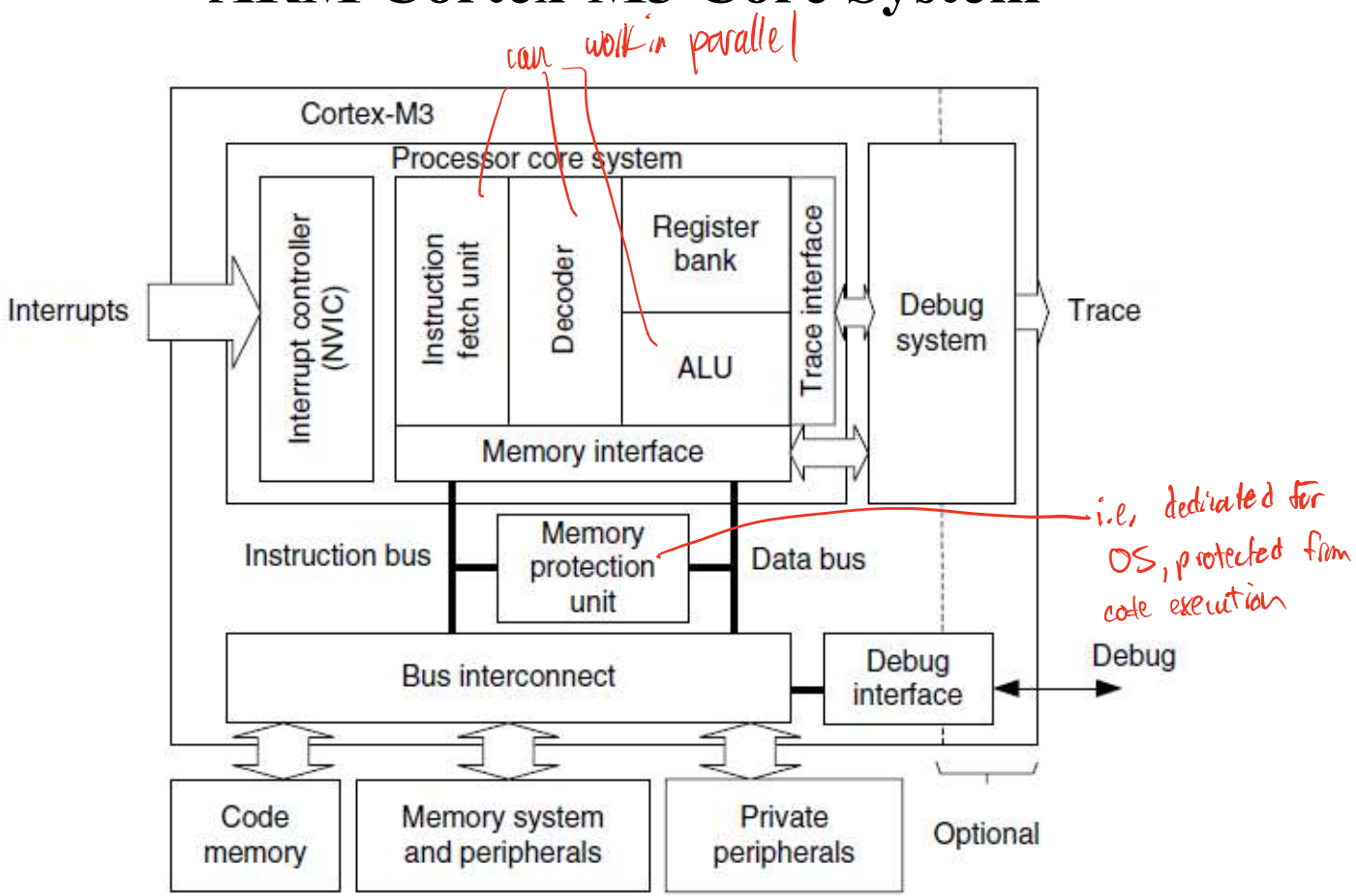
ARM Cortex-M optional components[6][7]

ARM Cortex-M	SysTick Timer	Bit-banding	Memory Protection Unit (MPU)	Tightly-Coupled Memory (TCM)	CPU cache	Memory architecture	ARM architecture
Cortex-M0[1]	Optional*	Optional[9]	No	No	No[10]	Von Neumann	ARMv6-M
Cortex-M0+[2]	Optional*	Optional[9]	Optional (8)	No	No	Von Neumann	ARMv6-M
Cortex-M1[3]	Optional	Optional	No	Optional	No	Von Neumann	ARMv6-M
Cortex-M3[4]	Yes	Optional*	Optional (8)	No	No	Harvard	ARMv7-M
Cortex-M4[5]	Yes	Optional*	Optional (8)	No	Possible[11]	Harvard	ARMv7E-M
Cortex-M7	Yes	No	Optional (8 or 16)	Optional	Optional	Harvard	ARMv7E-M

ARM Cortex-M3



ARM Cortex-M3 Core System



(i.e no address needs to be provided with int) Nonvectored interrupt controller

ARM Cortex-M3 Core System

NVIC - Interrupt controller (will get back to this in interrupt section).
Memory Protection Unit (MPU) - invokes rules for accessing memory; - Prevents the user from accessing system memory.

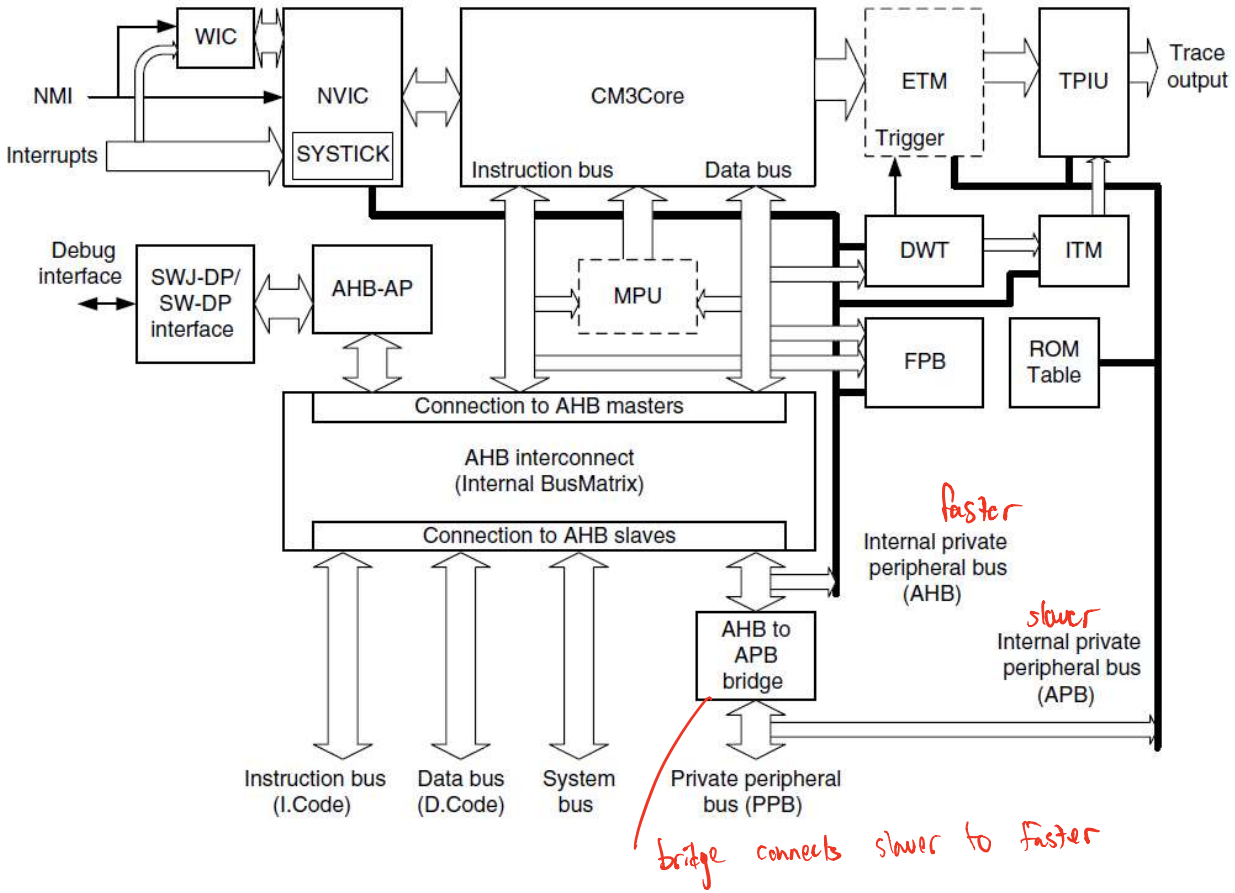
SYSTICK - countdown timer, used to generate interrupts. - needed for RTX

WIC - unit for waking up the CPU
ROM - small lookup table that stores configuration information

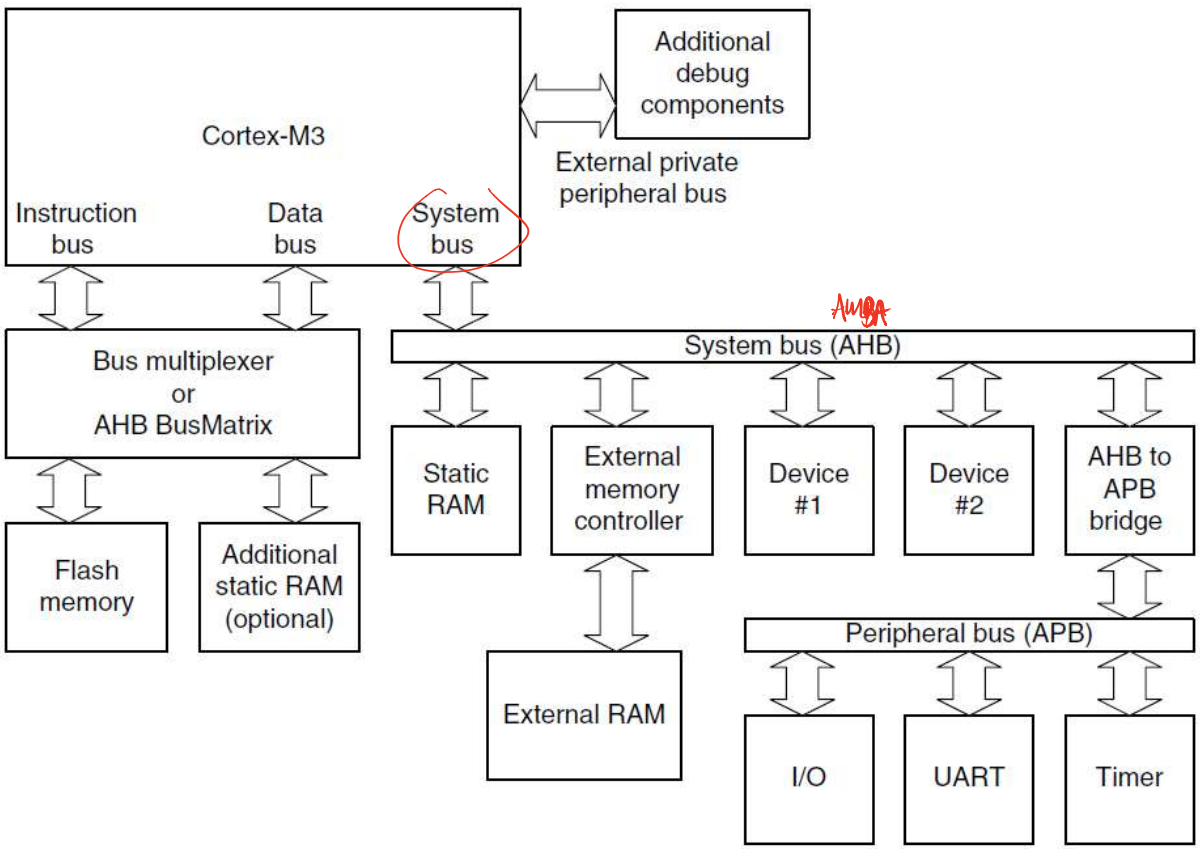
AMBA (discussed in SoC course)

BusMatrix - interconnect used to transfer data on different busses simultaneously.
Rest are **debugging** blocks ----> SW-DP (Serial Wire Debug Port), ETM (Embedded Trace Macrocell), DWT (Data Watch-point and Trace), ITM (Instrumentation Trace Macrocell), etc.

Cortex-M3 CPU Overview



Cortex-M3 Bus System



ARM Cortex-M3 Core System

advanced microcontroller bus architecture

- ARM uses the AMBA bus
- AMBA is an open standard on-chip interconnect specification (i.e. it has protocols, handshake methods, etc.)

Cortex-M3 uses the following AMBA based busses:

APB 1) Advanced Peripheral Bus - low bandwidth, used for processor to peripheral transactions

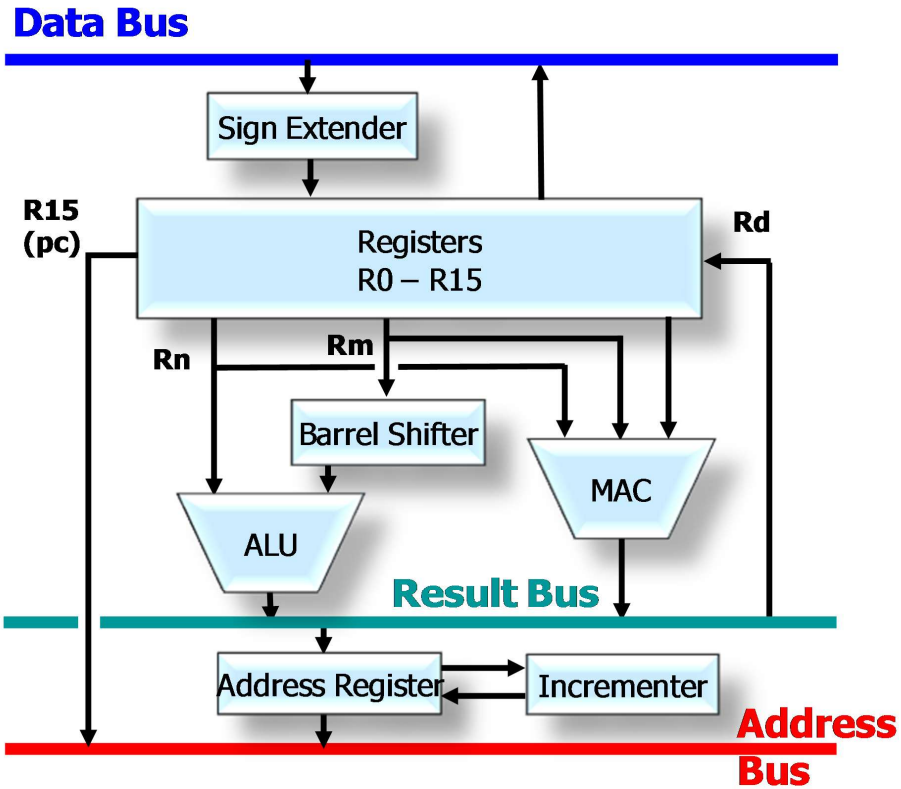
AHB 2) ARM High-Performance Bus - high bandwidth/data-width transfers for high performance, uses bursts

Cortex M3 has a Harvard architecture CPU: there are dedicated instruction and data busses.

System bus ⇒ AMBA

ARM Cortex-M3

Introduced in 2004, the mainstream ARM processor developed specifically with microcontroller applications in mind.



2's complement.
hardware sign extender.

ARM Cortex-M3

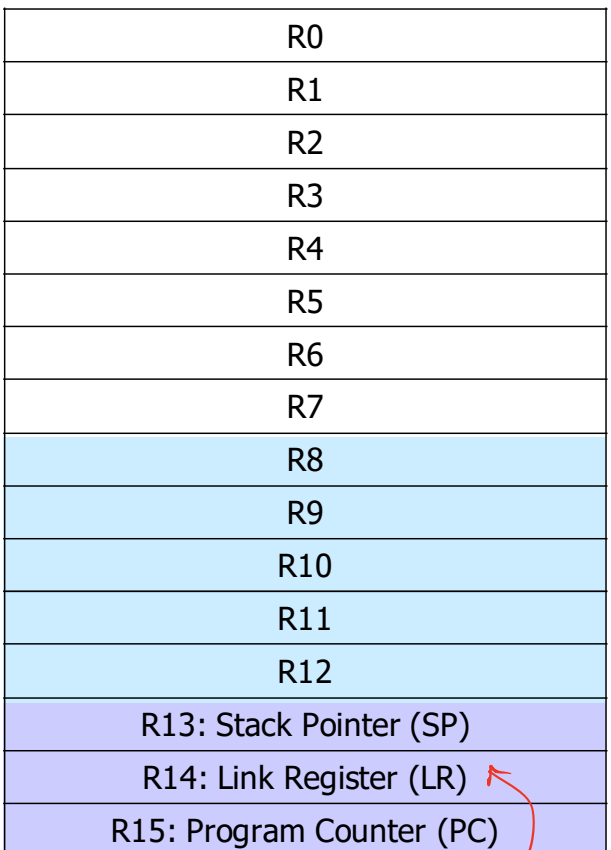
- Implement Thumb-2 instruction subset of ARM Instruction Set.
- Most Thumb-2 instructions are 16-bit wide that are expanded internally to a full 32-bit ARM instructions.
- ARM CPUs are capable of performing multiple low-level operations in parallel.
- A hardware sign extender convert 8-16 bit operands to 32-bit
- Load store architecture.
- Barrel shifter allows operand R_m to be shited first and then ALU can perform another operation (e.g. add, subtract, mul etc.)

- MAC is memory address calculator for different addressing of arrays and repetitive address calculations.
- R_0-R_{12} GPR, R_{13}-R_{15} special purpose registers i.e. SP, PC and LR (that holds the return address when a subroutine is called.

ARM Registers

ARM Mode:

15 general purpose registers



Thumb Mode:

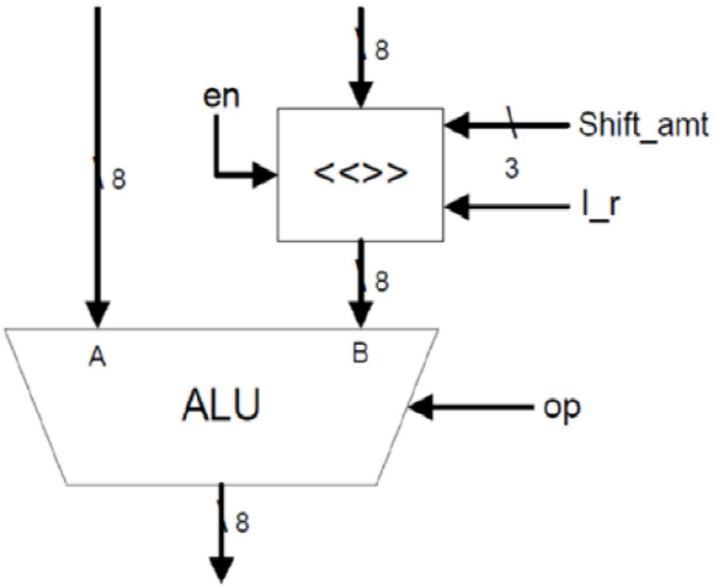
8 general purpose registers

7 "high" registers

r8-R12 only accessible with MOV, ADD, or CMP

useful when returning from subroutines

Barrel Shifting



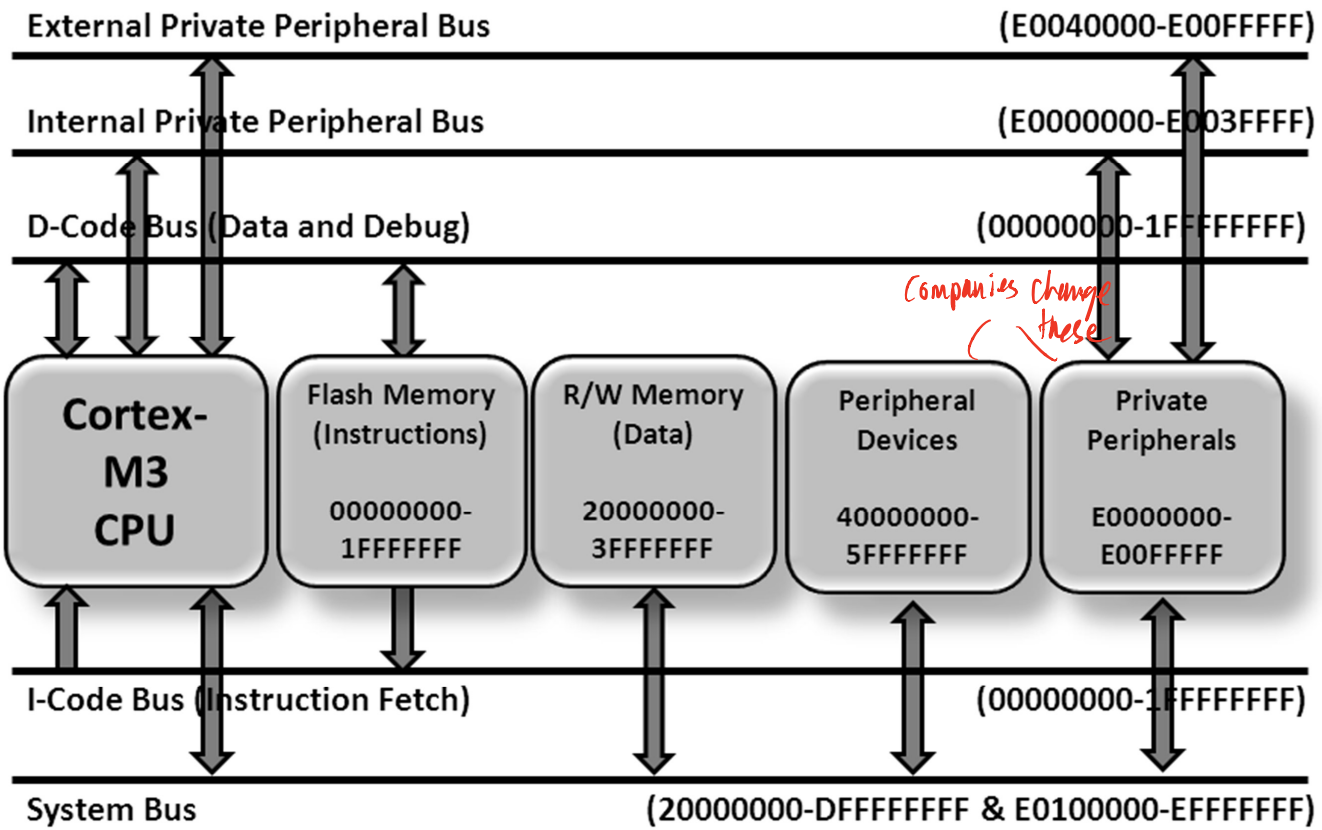
- Barrel shifter rotates/shift instruction operand prior to inputting the value into the ALU

one operation in one clock.

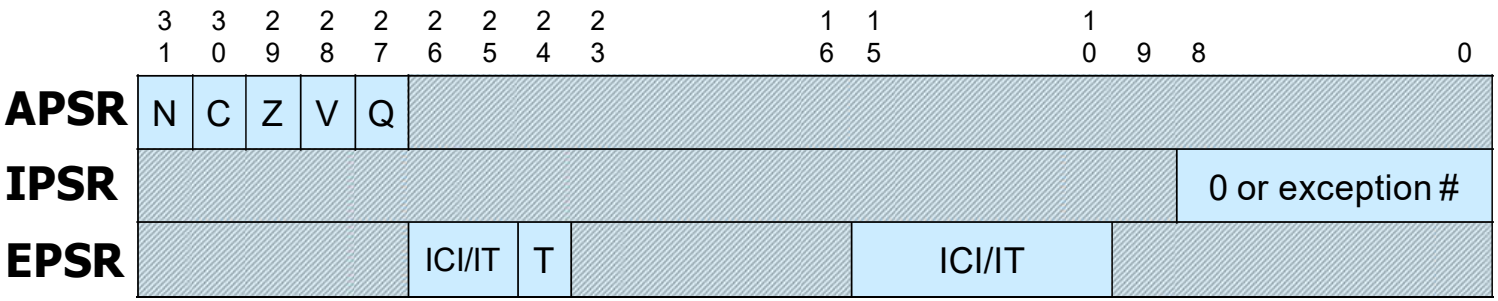
MUL R1 R2 #2 (LSL R1 R2 #2)
ADD R5, R1, R4

ADD R5, R4, R2, LSL #2

ARM Cortex-M3 Bus



Status Registers (xPSR)



Bits	Name	Description
31	N	Negative (bit 31 of result is 1)
30	C	Unsigned Carry
29	Z	Zero or Equal
28	V	Signed Overflow

Most important for application programming

PSR: Program Status Register

Divided into three bit fields

- Application Program Status Register (APSR)
- Interrupt Program Status Register (IPSR)
- Execution Program Status Register (EPSR)

All one register

IPSR holds the exception number for exception processing.

EPSR has the following useful bits.

- ICI/IT bits hold the state information for IT block instructions or instructions that are suspended during interrupt processing.
- Q-bit is the sticky saturation bit and supports two rarely used instructions (SSAT and USAT)

SSAT{cond} Rd, #sat, Rm{, shift} ARM7 doesn't have this

SSAT: Saturate Instruction

- Consider two numbers 0xFFFF FFFE and 0×0000 0002. A 32-bit mathematical addition would result in 0×1 0000 0001 which contain 9 hex digits or 33 binary bits. If the same arithmetic is done in a 32-bit processor, ideally the carry flag will be set and the result in the register will be 0×0000 0001.
- If the operation was done by any comparison instruction this would not cause any harm but during any addition operation this may lead to un-predictable results if the code is not designed to handle such operations. Saturate arithmetic says that when the result crosses the extreme limit the value should be maintained at the respective maximum/minimum (in our case result will be maintained at 0xFFFF FFFF which is the largest 32-bit number).
- Saturate instructions are very useful in implementing certain DSP algorithms like audio processing.

- Also a new flag field called 'Q' has been added to the ARM processor to show us if there had been any such saturation taken place or the natural result itself was the maximum.

ARM Operating Modes & Register Usage

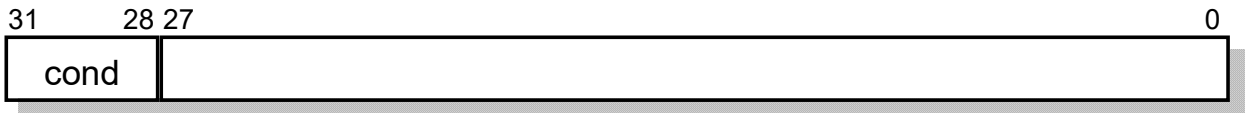
CPSR[4:0]	Mode	Use	Registers
10000	User	Normal user code	user
10001	FIQ	Processing fast interrupts	_fiq
10010	IRQ	Processing standard interrupts	_irq
10011	SVC	Processing software interrupts (SWIs)	_svc
10111	Abort	Processing memory faults	_abt
11011	Undef	Handling undefined instruction traps	_und
11111	System	Running privileged operating system tasks	user

Exception vector addresses

Exception	Mode	Vector address
Reset	SVC	0x00000000
Undefined instruction	UND	0x00000004
Software interrupt (SWI)	SVC	0x00000008
Prefetch abort (instruction fetch memory fault)	Abort	0x0000000C
Data abort (data access memory fault)	Abort	0x00000010
IRQ (normal interrupt)	IRQ	0x00000018
FIQ (fast interrupt)	FIQ	0x0000001C

Non-vectored
usually jump to some routine to check source of interrupt, then jump to actual routine.

The ARM Condition Code Field

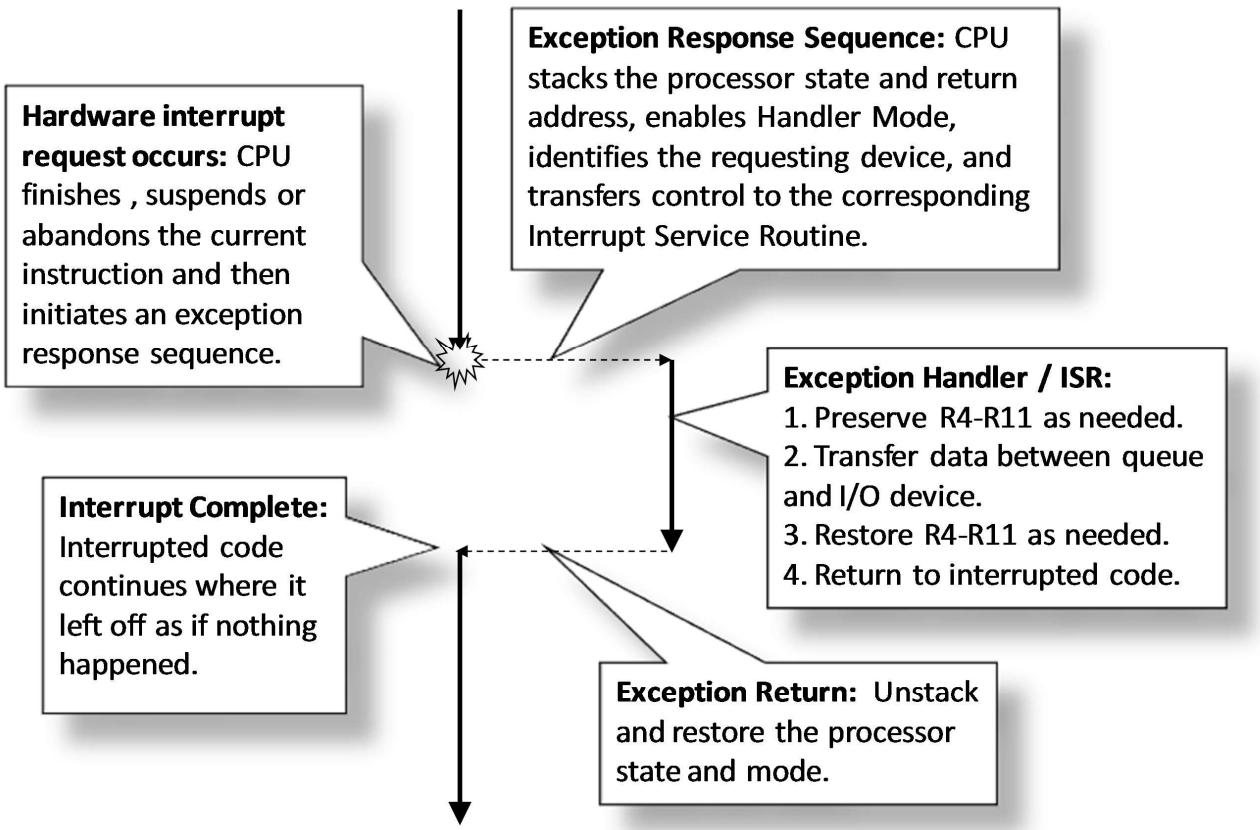


ARM condition codes

Opcode [31:28]	Mnemonic extension	Interpretation	Status flag state for execution
0000	EQ	Equal / equals zero	Z set
0001	NE	Not equal	Z clear
0010	CS/HS	Carry set / unsigned higher or same	C set
0011	CC/LO	Carry clear / unsigned lower	C clear
0100	MI	Minus / negative	N set
0101	PL	Plus / positive or zero	N clear
0110	VS	Overflow	V set
0111	VC	No overflow	V clear
1000	HI	Unsigned higher	C set and Z clear
1001	LS	Unsigned lower or same	C clear or Z set
1010	GE	Signed greater than or equal	N equals V
1011	LT	Signed less than	N is not equal to V
1100	GT	Signed greater than	Z clear and N equals V
1101	LE	Signed less than or equal	Z set or N is not equal to V
1110	AL	Always	any
1111	NV	Never (do not use!)	none

ARM - Interrupt Processing

(non-vectored)



Exception/Interrupt Handler

Exception: a condition that needs to halt the normal sequential flow of instruction execution.

Exception Categories: Reset, SVC Supervisor Call, Fault and Interrupts

Each exception has:

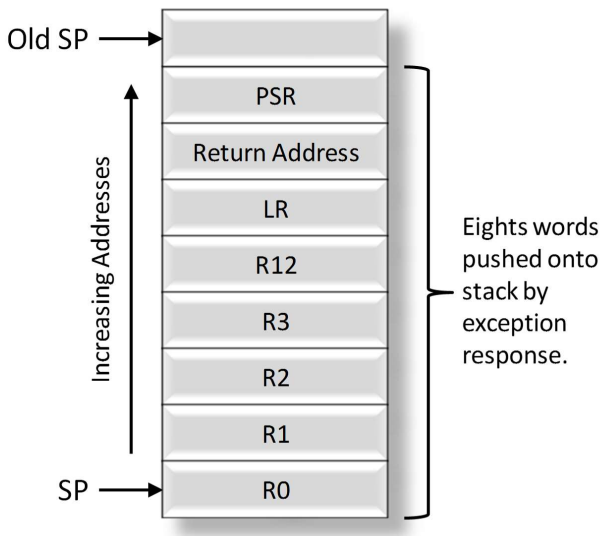
- An exception number
- A priority level
- An exception handler routine
- An entry in the vector table

Exception Response

- Processor state (8 words) stored on stack: CPSR, Return Address, LR, R12, R3 - R0.
- Processor switched (from Thread Mode) to Handler Mode
- PC ← vector table [exception #]

Exception Handlers and Return

An exception handler (ISR) is a software routine that is executed when a specific exception condition occurs.



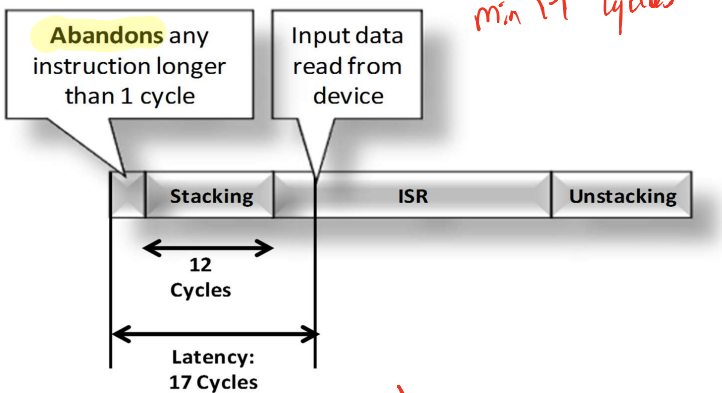
Exception return occurs when in Handler Mode and one of the following instructions is executed:

- POP/LDM includes the PC, or
- LDR with PC as destination or
- BX with any register as the source

Interrupt Stacking

Interrupt Latency

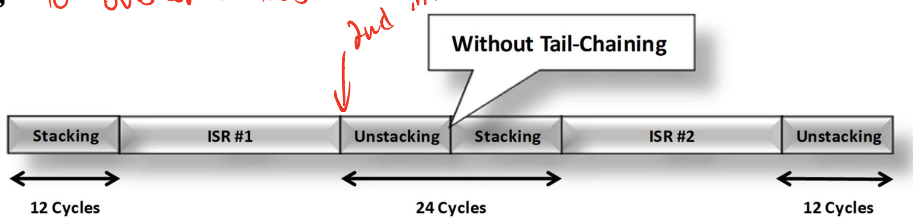
min 17 cycles for ISR to start.

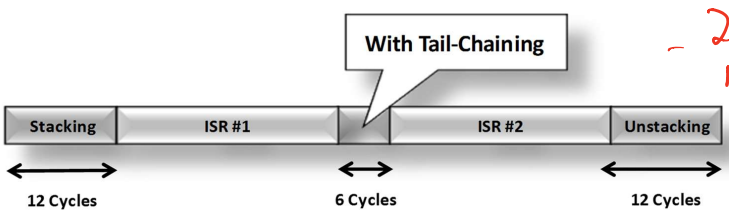


Tail Chaining

to overcome this

(Multiple Interrupts)





- 2nd interrupt serviced much faster (6 cycles)

Interrupt Latency Reduction

Time from interrupt request to the corresponding interrupt handler begins to execute.

1. Suspend or Abandon Instruction Execution:
No need to suspend single cycle instruction but multiple cycle ones.

2. Late Arrival Processing:
CPU has begun an interrupt response sequence and another high priority interrupt arrive during the stacking operation. – will go for higher priority one first.

3. Tail Chaining:
In most CPUs when two ISRs execute back to back, the state information is popped off the stack at the end of 1st interrupt only to be pushed back at the beginning of the 2nd interrupt.
M3 completely eliminates this useless pop-push sequence with a technique called tail-chaining, lowering the ISR transition time from 24 down to 6 clock cycles.

```
CPSIE  i  ; Enable External Interrupts
CPSID  i  ; Disable External Interrupts
```

M3 (Interrupt/Exception) Vector Table

Exception Type	Position	Priority	Comment
	0	-	Initial SP value (loaded on reset)
Reset	1	-3	Power up and warm reset
NMI	2	-2	Non-Maskable Interrupt
Hard Fault	3	-1	
Memory Mgmt	4	Settable	
Bus Fault	5		Address/Memory-related faults
Usage Fault	6		Undefined instruction
	7-10		*Reserved*
SVCall	11		Software Interrupt (SVC instruction)
Debug Monitor	12		
	13		*Reserved*
PendSV	14		
SysTick	15		System Timer Tick
Interrupts	≥16		External; fed through NVIC

Nested Vectored Interrupt Controller

Mapped to addresses $E000E100\text{-}E000ECFF_{16}$

It provides ability to:

- Individually Enable/Disable interrupts from specific devices.
- Establishes relative priorities among the various interrupts.

NVIC INTERRUPTS

0-4	GPIO Ports A-E
5,6	UART 0 & 1
7	SSI
8	I^2C
9	PWM Fault
10-12	PWM Generator 0-2
13	Reserved
14-17	ADC Sequence 0-3
18	Watchdog Timer
19-24	Timer 0a-2b
25	Analog Comparator
26-27	Reserved
28	System Control
29	Flash Control
30-31	Reserved

Bit in the interrupt registers