ARM
(Introduction & Architecture)

What is an Embedded System?

Embedded System=Hardware + Software

Definition: It is a computational engine, employing hardware and software, designed to perform specific function/s.

The software is used for providing features and flexibility.

The hardware is used for performance and sometimes security.

ARM Embedded Systems

- Key component of many 32 – bit embedded systems
- Portable Consumer devices
- ARM1 prototype in 1985
- One of the ARM's most successful cores is the ARM7TDMI,provides high code density and low power consumption.

The RISC Design Philosophy

- ARM Core uses a RISC architecture
- ARM licenses its cores out and other companies make processors based on its cores

The RISC Design Philosophy

- RISC is characterized by limited number of instructions
- A complex instruction is obtained as a sequence of simple instructions.so,in RISC processor software is complex but the processor architecture is simple.
- Large number of registers are required.
- Pipelined instruction execution.

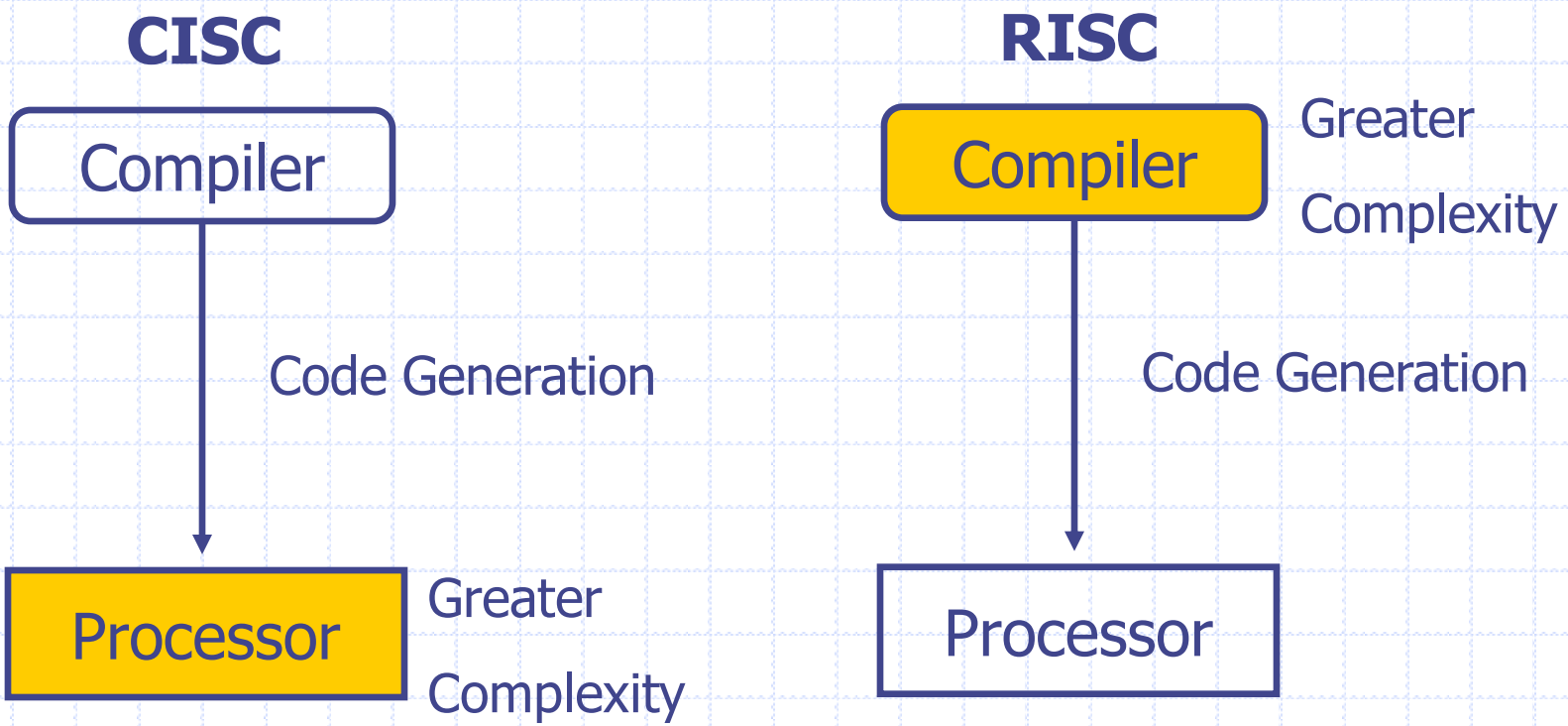
  Ex : ARM, ATMEL AVR, MIPS, Power PC etc

The CISC Design Philosophy

- CISC is characterized by large instruction set.
- The aim of designing CISC processors is to reduce software complexity by increasing the complexity of processor architecture.
- Very small number of registers are available.

  Ex : Intel X86 family,Motorola 68000 series.

CISC vs. RISC


CISC
Compiler
Code Generation
Processor
Greater
Complexity
RISC
Compiler
Greater
Complexity
Code Generation
Processor

RISC – 4 major design rules

- Instructions
  - Reduced Number of Instructions
  - Execute in a single cycle
  - The compiler synthesizes complicated operations
  - Each instruction is a fixed length

RISC – 4 major design rules

- Pipelines
  - The processing of instructions is broken down into smaller units that can be executed in parallel by pipelines
  - Pipeline advances by one step on each cycle for maximum throughput

RISC – 4 major design rules

- Registers
  - Have a large general purpose register set
  - Any register can contain either data or address
  - CISC has dedicated registers for specific purposes.

RISC – 4 major design rules

- Load – Store Architecture
  - Separate load and store instructions transfers data between the register bank and external memory

The ARM Design Philosophy

- Reduce power consumption
- High code density
- Price sensitive
- Reduce the area of the die taken up by the embedded processor
- Incorporated hardware debug technology

Instruction set for Embedded Systems

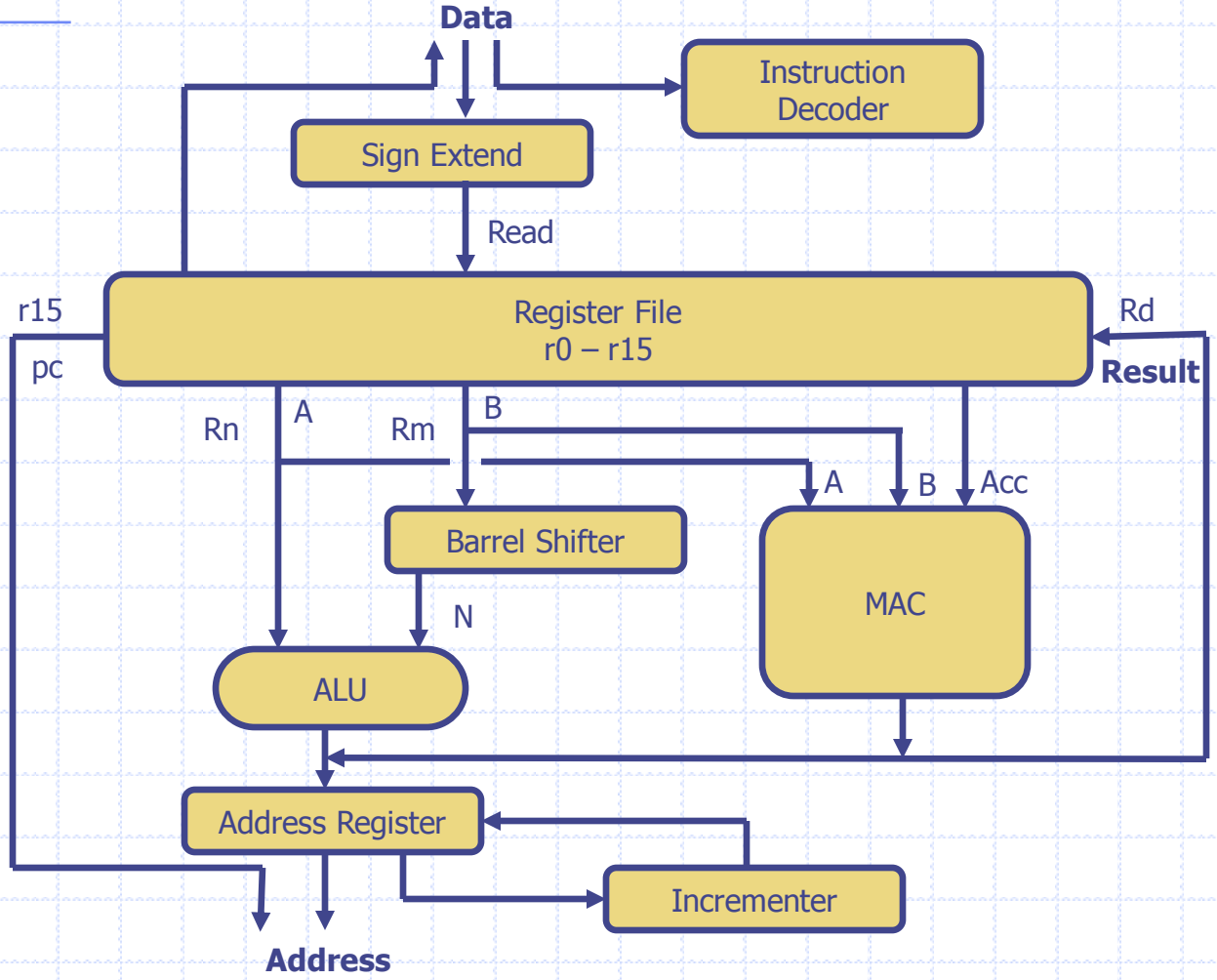
- Variable cycle execution for certain instructions
- Inline barrel shifter leading to more complex instructions
- Thumb 16 – bit instructions
- Conditional execution
- Enhanced Instructions

ARM Processor Fundamentals

Agenda

- Registers
- CPSR
- Pipeline
- Exceptions, Interrupts and the Vector Table
- Core Extensions
- Architecture Revisions
- ARM Processor Families
- Summary

ARM core dataflow model

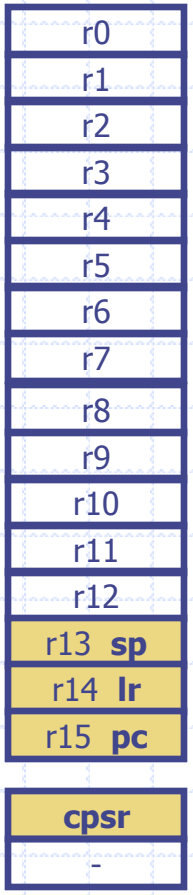


ARM core dataflow model

- Functional units connected by data buses
- **Data Bus** → Data or Instruction
- Von Neumann architecture → Data and instruction share the same bus
- Load Store Architecture
- Register File – 32 bit registers
- ARM instruction has 2 source and 1 destination register
- L/S inst: use the ALU to generate an address to be held in the address reg. and broadcast on the **Address Bus**
- **Result Bus**

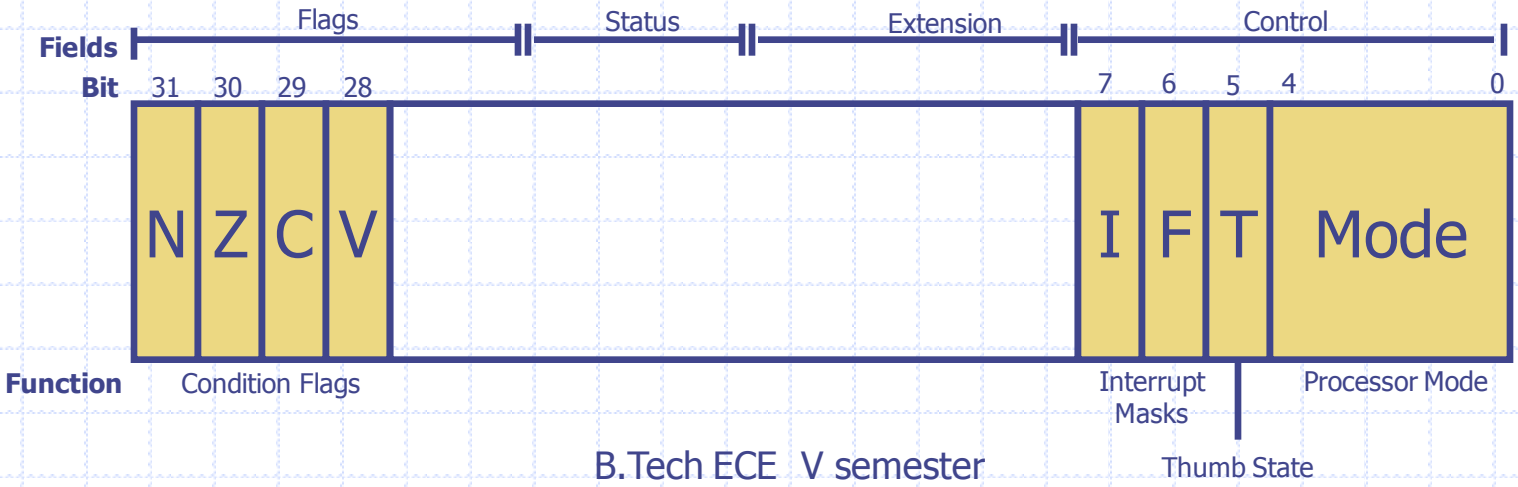
Registers – User Mode

- 32 bit in size
- Hold either data or address
- 16 data registers(r0 – r15) and 2 processor status register (cpsr & spsr)
- r13, r14, r15 – Special functions
- r13 (sp) –stores the head of the stack in the current processor mode
- r14 (lr) – the core puts the return address whenever it calls a subroutine
- r15 (pc) – contains the address of the next instruction to be fetched by the processor
- Which register are visible to the programmer depend upon the current mode of the processor



Current Program Status Register

- To monitor and control internal operations
- Some ARM Processor core have extra bits allocated



Processor Modes

- Determines which registers are active and the access rights to the cpsr register itself
- Privileged & Nonprivileged
  - Abort
  - Fast Interrupt Request
  - Interrupt Request
  - Supervisor
  - System
  - Undefined
  - User

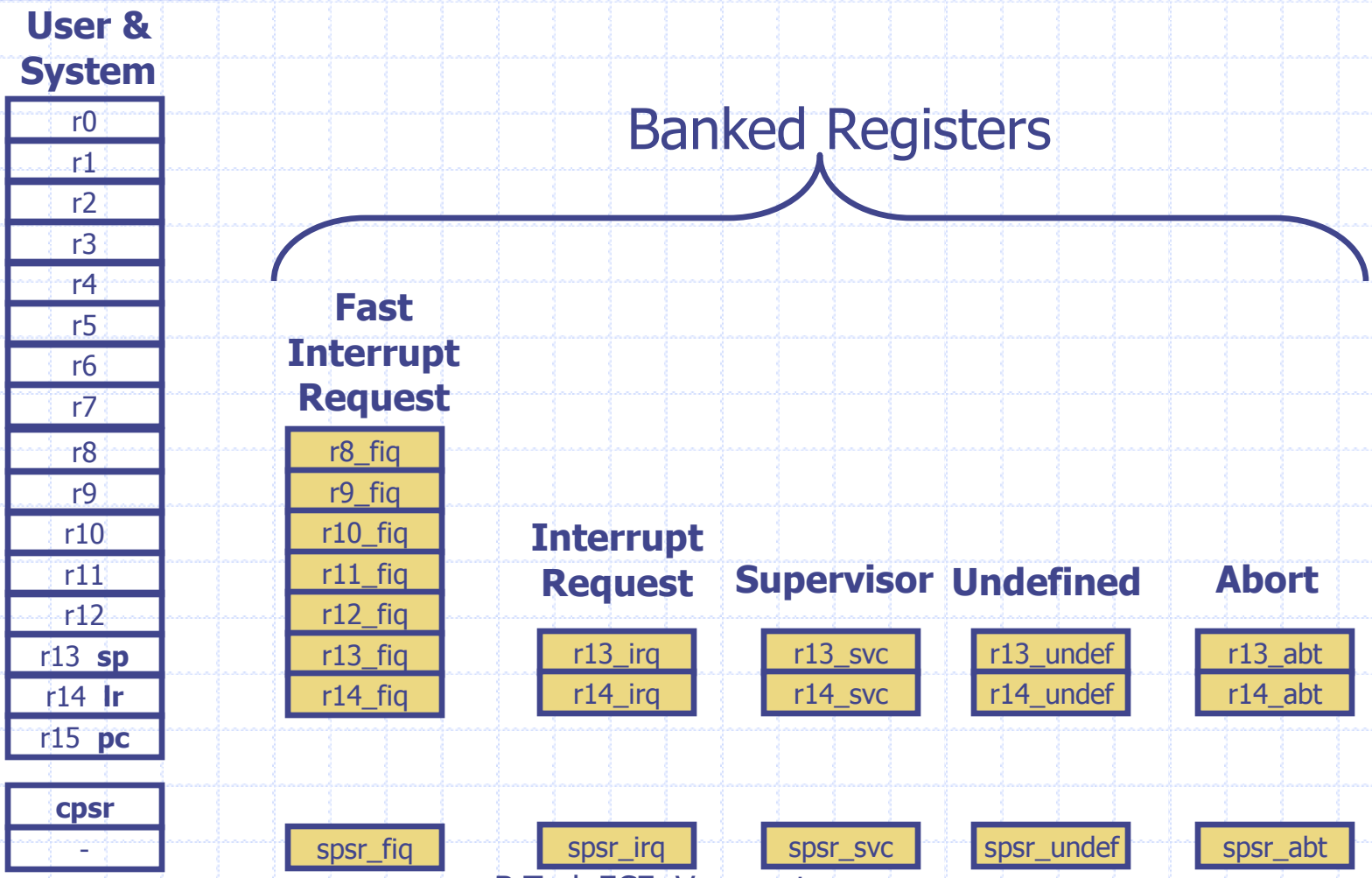
Privileged-R/W access to CPSR

Nonprivileged-R Access to CF,R/W access to ConditionFlags

Processor Modes

- Abort – Failed attempt to access memory
- FIQ IRQ – Two interrupt levels available on the ARM processor
- Supervisor – OS kernel operates
- System – Special version of user mode that allows full R/W access to the cpsr
- Undefined – processor encounters an instruction that is undefined
- User – used in programs & applications
- When a power is applied to the core it starts in supervisor mode.

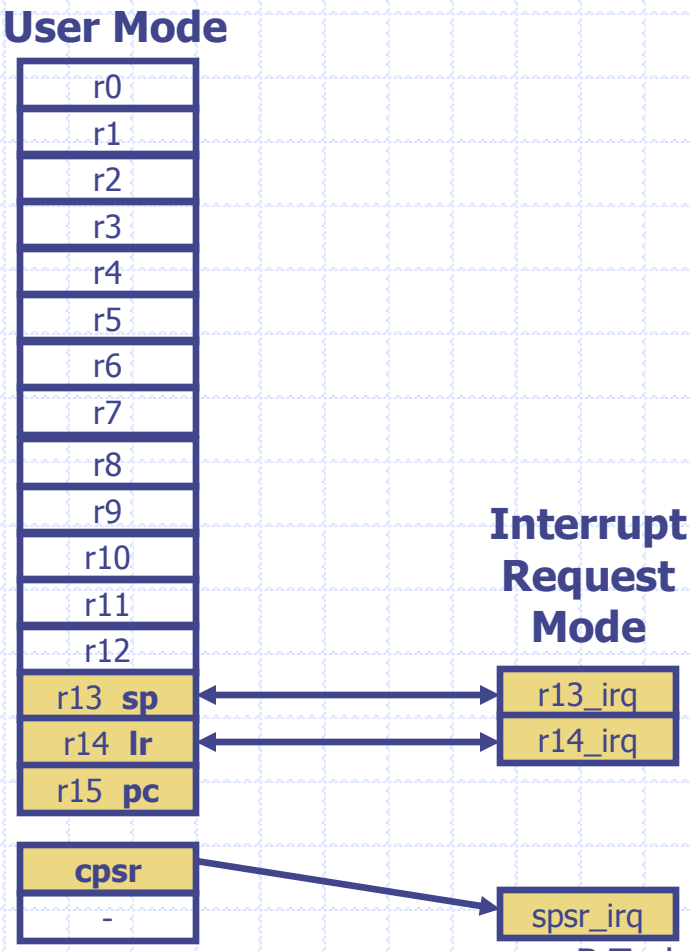
Banked Registers



Banked Registers

- Banked registers are available only when the processor is in a particular mode
- Every processor mode *except user mode* can change mode by writing directly to the mode bits of the cpsr
- Banked registers are a subset of the main 16 registers
- If we change processor mode, a banked register from the new mode will replace an existing register
- Exceptions and Interrupts cause a mode change

Changing mode on an exception



- This change causes user register r13 and r14 to be banked
- The user registers are replaced with registers r13_irq and r14_irq
- spsr stores the previous mode cpsr

Processor Mode

Mode	Abbr:	Privileged	Mode[4:0]
Abort	abt	yes	10111
Fast Interrupt Request	fiq	yes	10001
Interrupt Request	irq	yes	10010
Supervisor	svc	yes	10011
System	sys	yes	11111
Undefined	und	yes	11011
User	usr	no	10000

cpsr is not copied into the ***spsr*** when a mode change is forced due to a program writing directly to the ***cpsr***.

State and Instruction Sets

- There are three instruction sets
  - ARM
  - Thumb
  - Jazelle

The Jazelle instruction set is a closed instruction set and is not openly available.

To take advantage of Jazelle extra software has to be licensed from both ARM Limited and Sun Microsystems.

State and Instruction Sets

	ARM (cpsr T = 0)	Thumb (cpsr T = 1)
Instruction Size	32 bit	16 bit
Core Instruction	58	30
Conditional Execution	Most	Only branch instructions
Data Processing Instructions	Access to barrel shifter and ALU	Separate barrel and ALU instructions
Program Status Register	R/W in privileged mode	No direct access
Register Usage	15 GPR + PC	8 GPR + 7 high registers + PC

State and Instruction Sets

	Jazelle (cpsr T = 0, J = 1)
Instruction Size	8 bit
Core Instruction	Over 60% of the java bytecodes are implemented in hardware; the rest of the codes are implemented in software

Interrupt Masks

- Are used to stop specific interrupt requests from interrupting the processor
  - IRQ
  - FIQ
- The I bit masks IRQ when set to binary 1, and F bit masks FIQ when set to binary 1

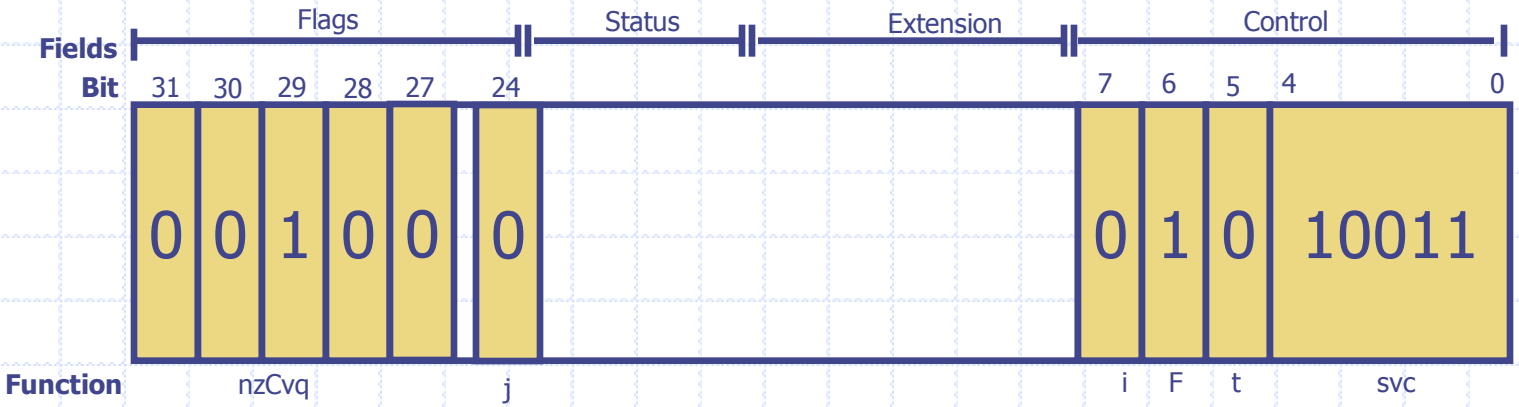
Condition Flags

Flag	Flag Name	Set when
Q	Saturation	The result causes an overflow and / or saturation
V	oVerflow	The result causes a signed overflow
C	Carry	The result causes an unsigned carry
Z	Zero	The result is zero, frequently used to indicate the equality
N	Negative	Bit 31 of the result is a binary 1

Condition Flags

- Condition flags are updated by comparisons and the result of ALU operations that specify the S instruction suffix
  - If SUBS results in a register value of zero, then the Z flag in the cpsr is set

Condition Flags – Eg



cpsr = nzCvqjiFt_SVC

Conditional Execution

- Conditional execution controls whether or not the core will execute an instruction
- Most instructions have a condition attribute that determines if the core will execute it based on the setting of the condition flags
- Prior to execution, the processor compares the condition attribute with the condition flags in the cpsr
- If they match, then the instruction is executed, otherwise the instruction is ignored
- When a condition mnemonic is not present, the default behaviour is set to always (AL) execute

Conditional Execution

Mnemonic	Name	Condition Flags
EQ	equal	Z
NE	not equal	z
CS HS	carry set/unsigned higher or same	C
CC LO	carry clear/unsigned lower	c
MI	minus/negative	N
PL	plus/positive or zero	n
VS	overflow	V
VC	no overflow	v
HI	unsigned higher	zC
LS	unsigned lower or same	Z or c
GE	signed greater than or equal	NV or nv
LT	signed less than	Nv or nV
GT	signed greater than	NzV or nzv
LE	signed less than or equal	Z or Nv or nV
AL	always (unconditional)	ignored

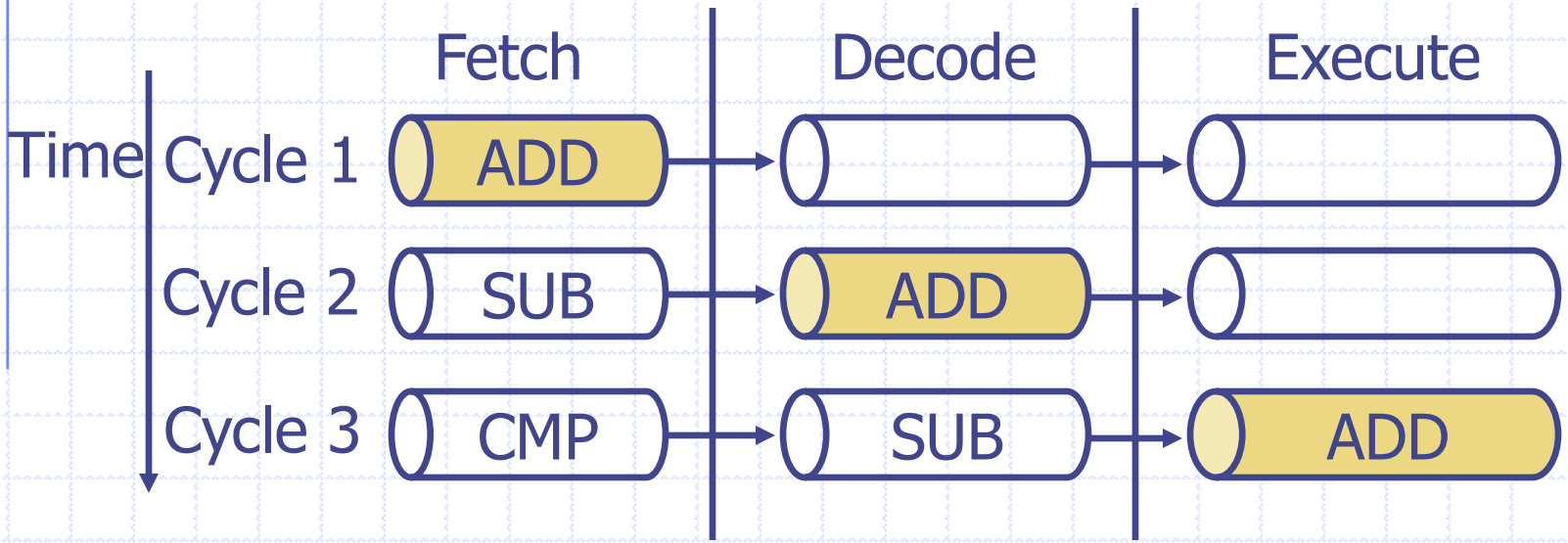
Pipeline

- Is a mechanism a RISC processor uses to execute instructions
- Using a pipeline speeds up execution by fetching the next instruction while other instructions are being decoded and executed

ARM7 Three stage pipeline

- Fetch loads an instruction from memory
- Decode identifies the instruction to be executed
- Execute processes the instruction and writes the result back to a register

Pipelined instruction sequence



- Filling the pipeline
- Allows the core to execute an instruction every cycle

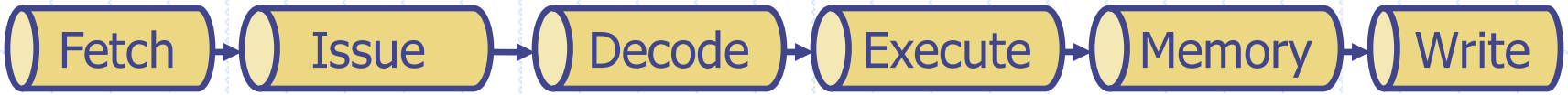
ARM9 Five stage pipeline

- Higher operating frequency → higher performance
- Latency increases
- Increase in instruction throughput by around 13% in 5 stage pipeline
- 1.1 Dhrystone MIPS per MHz

ARM9 Five stage pipeline

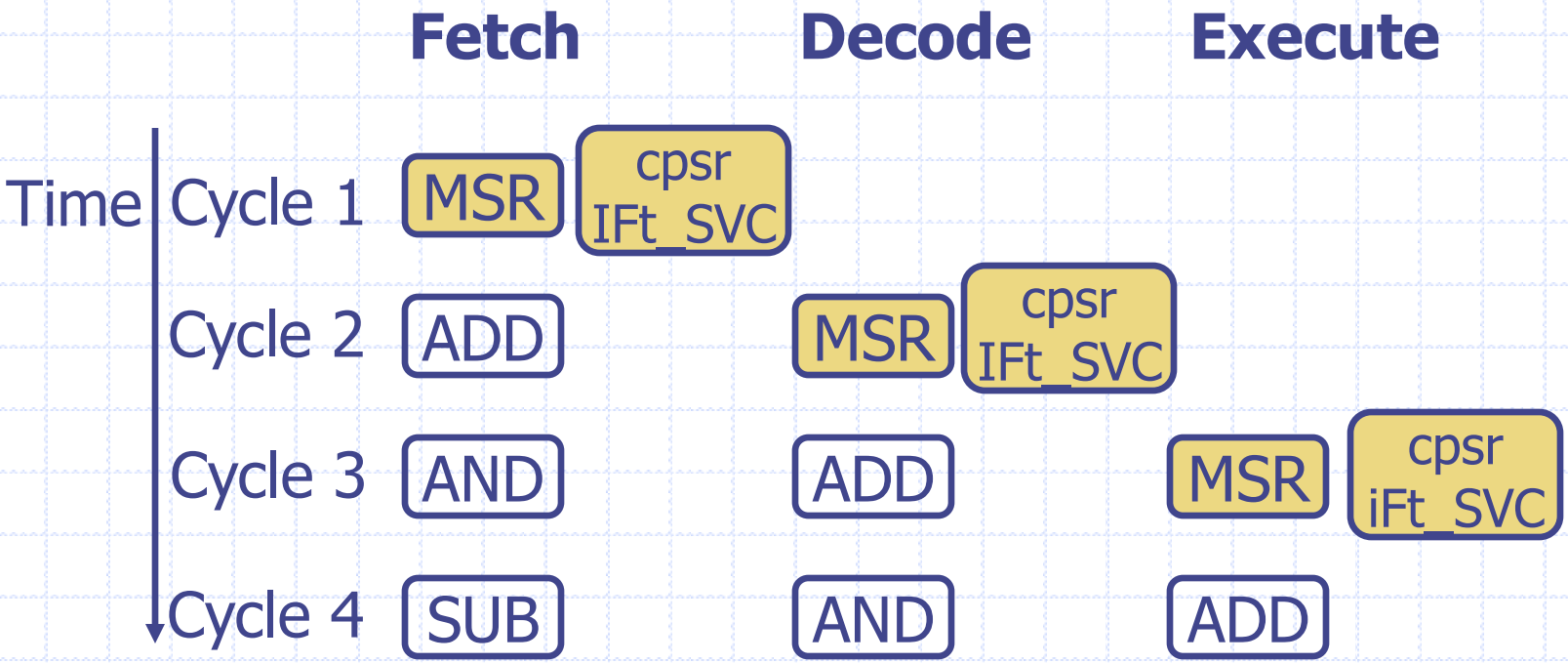
- Fetch
  - The instruction is fetched from memory and placed in the instruction pipeline
- Decode
  - The instruction is decoded and register operands read from the register file
- Execute
  - An operand is shifted and the ALU result generated
- Memory (Buffer/Data)
  - Data memory is accessed if required. Otherwise the ALU result is buffered for one clock cycle to give the same pipeline flow for all instructions
- Write (Write-Back)
  - The results generated by the instruction are written back to the register file, including any data loaded from memory

ARM10 Six stage pipeline



- Increase in instruction throughput by around 34% in 6 stage pipeline
- 1.3 Dhrystone MIPS per MHz
- Code written for the ARM7 will execute on ARM9 and ARM10

ARM Instruction Sequence



Pipeline Characteristics

- An instruction in the execute stage will complete even though an interrupt has been raised
- The execution of a branch instruction or branching by the direct modification of the PC causes the ARM core to flush its pipeline

Exceptions, Interrupts, and the Vector Table

- When an exception or interrupt occurs, the processor set the PC to a specific memory address
- The address is within a special address range called the vector table
- The entries in the vector table are instructions that branch to specific routines designed to handle a particular exception or interrupt
- When an exception or interrupt occurs,the processor suspends normal execution and starts loading instructions from the exception vector table.

Exceptions, Interrupts, and the Vector Table

Exception / Interrupt	Shorthand	Address	High Address
Reset	RESET	0x00000000	0xffff0000
Undefined Instruction	UNDEF	0x00000004	0xffff0004
Software Interrupt	SWI	0x00000008	0xffff0008
Prefetch Abort	PABT	0x0000000C	0xffff000C
Data Abort	DABT	0x000000010	0xffff0010
Reserved	-	0x000000014	0xffff0014
Interrupt Request	IRQ	0x000000018	0xffff0018
Fast Interrupt Request	FIQ	0x00000001C	0xffff001C

Exceptions, Interrupts, and the Vector Table

- **RESET** – when power is applied, branches to initialization code
- **UNDEF** – when the processor cannot decode an instruction
- **SWI** – when a SWI instruction is called
- **PABT** – attempts to fetch an instruction from an address without the correct access permissions
- **DABT** –attempts to access data memory without the correct access permissions
- **IRQ** – by external hardware
- **FIQ** – by external hardware requiring faster response time

Core Extensions

- Standard components placed next to the ARM core
- Improve performance, manage resources, provide extra functionality
- Three hardware extensions
  - Caches
  - Memory Management
  - Coprocessors

Caches

- Cache is a block of fast memory placed between main memory and the core
- Cache provides an overall increase in performance
- ARM has two forms of cache
  - Single unified cache for data and instruction
  - Separate caches for data and instruction

Memory Management

- MMU is a class of processor hardware components for handling memory accesses requested by the CPU.
- The functions of MMU's are
  - Translation of virtual address to physical address.
  - Memory protection
  - Cache control etc

Coprocessors

- Coprocessors can be attached to the ARM processor
- A separate chip,that performs lot of calculations for the microprocessor,relieving the CPU some of its work and thus enhancing overall speed of system.
- A secondary processor used to speed up operation by taking over a specific part of main processors work.
- The ARM processor uses coprocessor 15 registers to control cache, TCMs, and memory management

Architecture Revisions

- Every ARM processor implementation executes a specific instruction set architecture (ISA)
- ISA have more than one processor implementation

Nomenclature

- ARM{x}{y}{z}{T}{D}{M}{I}{E}{J}{F}{-S}
  - x - family
  - y – memory management / protection unit
  - z - cache
  - T – Thumb 16 – bit decoder
  - D – JTAG debug
  - M – fast multiplier
  - I – EmbeddedICE macrocell
  - E – Enhanced instructions (assumes TDMI)
  - J - Jazelle
  - F – vector floating point unit
  - S – Synthesizible version

Revision History

Revision	Example core Implementation	ISA enhancement
ARMv1	ARM1	First ARM Processor 26 – bit addressing
ARMv2	ARM2	32 – bit multiplier 32 – bit coprocessor support
ARMv2a	ARM3	On chip cache Atomic swap instruction Coprocessor 15 for cache management
ARMv3	ARM6 & ARM7DI	32 – bit addressing Separate cpsr & spsr New modes – UNDEF, ABORT MMU support – virtual memory
ARMv3M	ARM7M	Signed & unsigned long multiply inst.
ARMv4	StrongARM	Load – store instruction New Mode – system 26 bit addressing mode no longer supported

Revision History

Revision	Example core Implementation	ISA enhancement
ARMv4T	ARM7TDMI & ARM9T	Thumb
ARMv5TE	ARM9E & ARM10E	Superset of the ARMv4T Extra inst. added for changing state between ARM & Thumb Enhanced multiply instructions Extra DSP type instructions Faster multiply accumulate
ARMv5TEJ	ARM7EJ & ARM926EJ	Java acceleration
ARMv6	ARM11	Improved multiprocessor instructions Unaligned and mixed endian data handling New multimedia instructions

Description of cpsr

Parts	Bits	Architecture	Description
Mode	4:0	all	processor mode
T	5	ARMv4T	Thumb state
I & F	7:6	all	interrupt masks
J	24	ARMv5TEJ	Jazelle state
Q	27	ARMv5TE	condition flag
V	28	all	condition flag
C	29	all	condition flag
Z	30	all	condition flag
N	31	all	condition flag

ARM processor families

- ARM7, ARM9, ARM10 and ARM11
- 7, 9, 10, 11 indicate different core designs

ARM family attribute comparison

	ARM7	ARM9	ARM10	ARM11
Pipeline depth	three-stage	five-stage	six-stage	eight-stage
Typical MHz	80	150	260	335
mW/MHz	0.06 mW/MHz	0.19 mW/MHz (+ cache)	0. 5 mW/MHz (+ cache)	0.4 mW/MHz (+ cache)
MIPS/MHz	0.97	1.1	1.3	1.2
Architecture	Von Neumann	Harvard	Harvard	Harvard
Multiplier	8 x 32	8 x 32	16 x 32	16 x 32

ARM processor variants

CPU Core	MMU /MPU	Cache	Jazelle	Thumb	ISA	E
ARM7TDMI	none	none	no	yes	v4T	no
ARM7EJ-S	none	none	yes	yes	v5TEJ	yes
ARM720T	MMU	unified – 8K cache	no	yes	v4T	no
ARM920T	MMU	separate – 16K/16K D + I cache	no	yes	v4T	no
ARM922T	MMU	separate – 8K/8K D + I cache	no	yes	v4T	no
ARM926EJ-S	MMU	separate – cache TCM configurable	yes	yes	v5TEJ	yes
ARM940T	MPU	separate – 4K/4K D + I cache	no	yes	v4T	no
ARM946E-S	MPU	separate – cache TCM configurable	no	yes	v5TE	yes
ARM966E-S	none	separate – TCM configurable	no	yes	v5TE	no

ARM processor variants

CPU Core	MMU /MPU	Cache	Jazelle	Thumb	ISA	E
ARM1020E	MMU	separate – 32K/32K D + I cache	no	yes	v5TE	yes
ARM1022E	MMU	separate – 16K/16K D + I cache	no	yes	v5TE	yes
ARM1026EJ-S	MMU MPU	separate – cache TCM configurable	yes	yes	v5TE	yes
ARM1136J-S	MMU	separate – cache TCM configurable	yes	yes	v6	yes
ARM1136F-S	MMU	separate – cache TCM configurable	yes	yes	v6	yes

Cortex Family

- ARM Cortex-A Series - Application processors for complex OS and user applications
  - ARM Cortex-A8, ARM Cortex-A9
- ARM Cortex-R Series - Embedded processors for real-time systems
  - ARM Cortex-R4(F)
- ARM Cortex-M Series – Embedded processors optimized for cost sensitive applications, as Mobile devices
  - ARM Cortex-M0, ARM Cortex-M1, ARM Cortex-M3

Specialized Processors

- StrongARM
  - Digital Semiconductor + Intel
  - PDAs
  - Low power consumption
  - Harvard Architecture
  - 5 stage pipeline
  - No thumb support

Summary

- Data flow in an ARM core.
- 3 instruction sets
- Register file
- Extensions
  - Caches
  - Memory Management
  - Coprocessors
- ISA