Embedded Systems 1 (ENCE361)

Lecture 32
Instruction Set Architectures (ISAs)

A General Introduction

By: Dr. Steve Weddell

- Definitions & types
- The software/hardware interface
- Instruction set architectures types
- Load instruction example
- Data processing instruction example

Review of some key points…

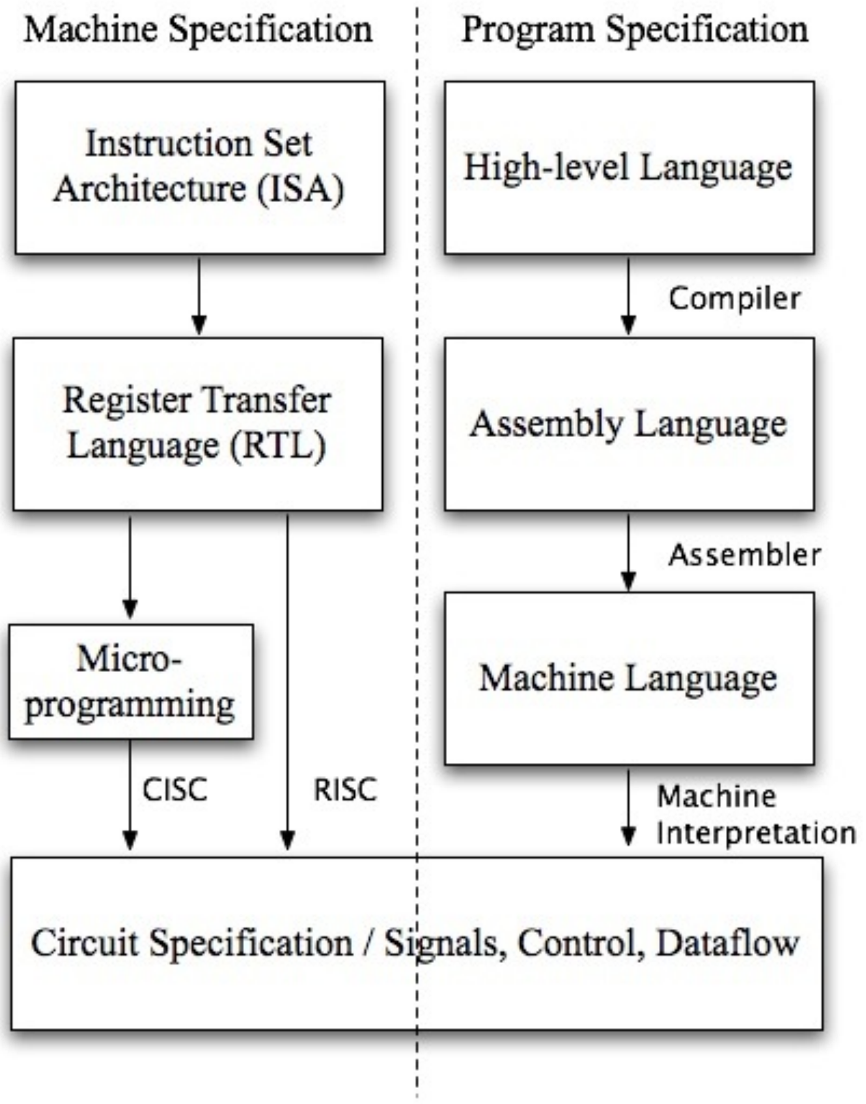
- Large digital systems use a modular, hierarchical design approach.
- Most complex digital systems, such as CPUs, are partitioned into two distinct modules:
  - The ***datapath*** which performs data processing operations and,
  - the ***control unit*** that provides the *sequencing* of operations.
- The control unit sends control signals to the datapath.
- The control unit receives status information from the datapath via ***status bits (***e.g Z, V) located in the *condition code register*.
- Status bit information is used by the control unit to aid in the sequencing of operations.

Instruction Set Architecture - ISA

A formal definition of ISA:

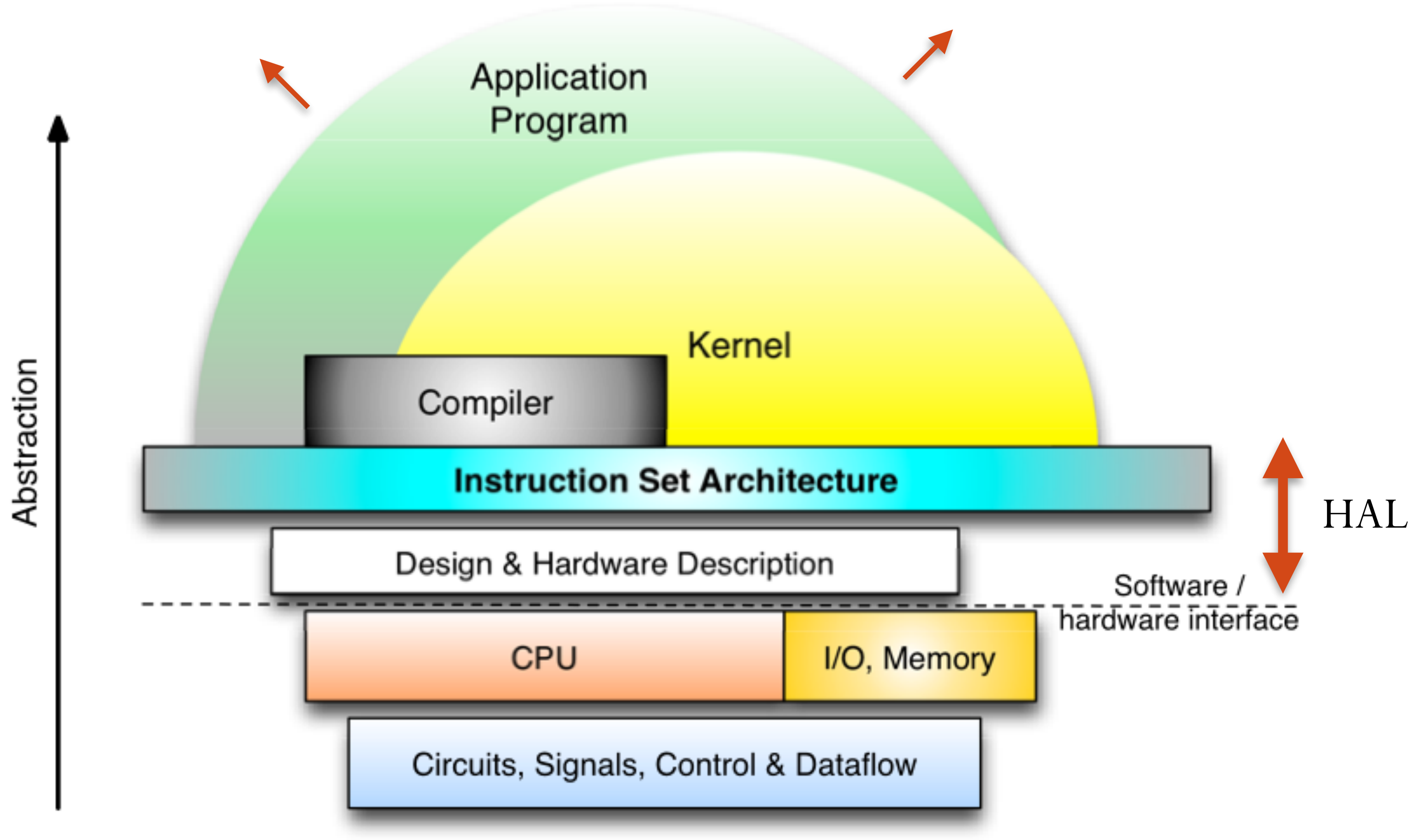
Instruction Set Architecture refers to how a programmer would view an instruction set. This would encompass the function of each instruction, how data operands can be used by each instruction (addressing modes) and the CPU, memory, and IO operations that are required by the execution of instructions.

Understanding the hardware software interface…



- *Program specification* represents a top-down program (software) abstraction.
- *Machine specification* represents a top-down machine (hardware / firmware) abstraction.
- Reduced instruction set computers (RISC) use hardware state machines for control.
- Complex instruction set computers (CISC) use software (micro-coded) state machines for control.

Instruction Set Architecture - ISA


Application Program
Kernel
Compiler
Instruction Set Architecture
Design & Hardware Description
CPU
I/O, Memory
Circuits, Signals, Control & Dataflow
Abstraction
HAL
Software / hardware interface

HAL: hardware abstraction layer

Types of Instruction Set Architectures

- Most computations require two sources of data (operands), and a destination (operand) to store the result. For example, if A & B were source registers and D a destination register, then, in <u>register transfer language</u> (RTL): D ← A *operation* B.
- The number of operands an instruction will support can vary and is related to the number of address and data buses that the architecture will provide. These are summarised as:
  - ***Three Address Instructions***: Two operand sources and one destination are included in the instruction. e.g. ADD R3,R1,R2 (in RTL: R3 ← R2 + R1), MIPS, ARM. Used in Register-to-Register (load-store) architectures;
  - ***Two Address Instructions***: One source and one destination are explicitly stated in the instruction. The second source either doubles as the destination or is implied in the instruction. e.g. ABA ; (A ← A+B), CPU-12 by Freescale;
  - ***One Address Instructions***: Uses <u>an implied operand</u> to double as a source address as well as a combined destination address. A common architecture for *single accumulator* CPUs.

    e.g., ADD #$F0 (Acc ← Acc + $F0) M68HC05 ; One accumulator (Acc), only, is supported. Alternatively, ADD $F0. What's the difference between these?
  - ***Zero Address Instruction***: All operations on operands are assumed to be ordered in a specific area of memory known as the stack. e.g., PSHA, PSHB, ADD, PULA etc.

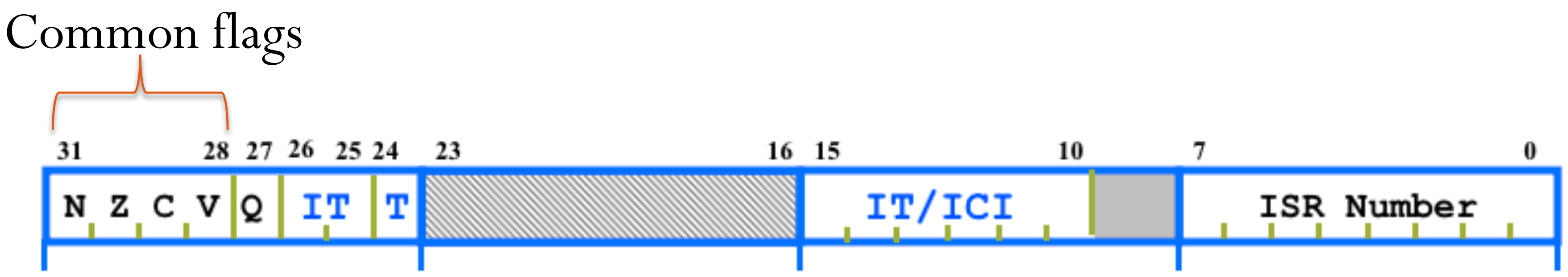
Some terms used to describe ISAs

1. An ***operand*** is additional information that an instruction may require to complete its operation, and is distinct from an instruction's ***operation*** (op) code. Operands can be a literal value, an address, contents of a register, or a specific register.
2. The location of the argument given in terms of an instruction's operand(s) and addressing mode, is referred to as the ***effective address***. This can be source or destination (see Slide 9).
3. A CPU ***Datapath*** comprises circuit modules that provide access to the data (operands) to be executed by each instruction, e.g., the ***Register file***.
4. A CPU ***control unit*** is effectively a state machine, designed to execute each instruction stage in a specific sequence e.g:
    A. ***Fetch, Decode***, ***Execute*** (in terms of implicit instructions)
    B. ***Fetch, Decode, Read Op, Execute, Store Result*** (in terms of instructions that support explicit operands).
5. The ***Condition Code Register*** (CPSR in ARM processors) contains a series of single-bit flags and is used to provide the Control Unit with information about the Datapath e.g., register R0 used in the last operation is zero. BNE, CMP, etc., instructions will **test** such bits and output control signals accordingly.

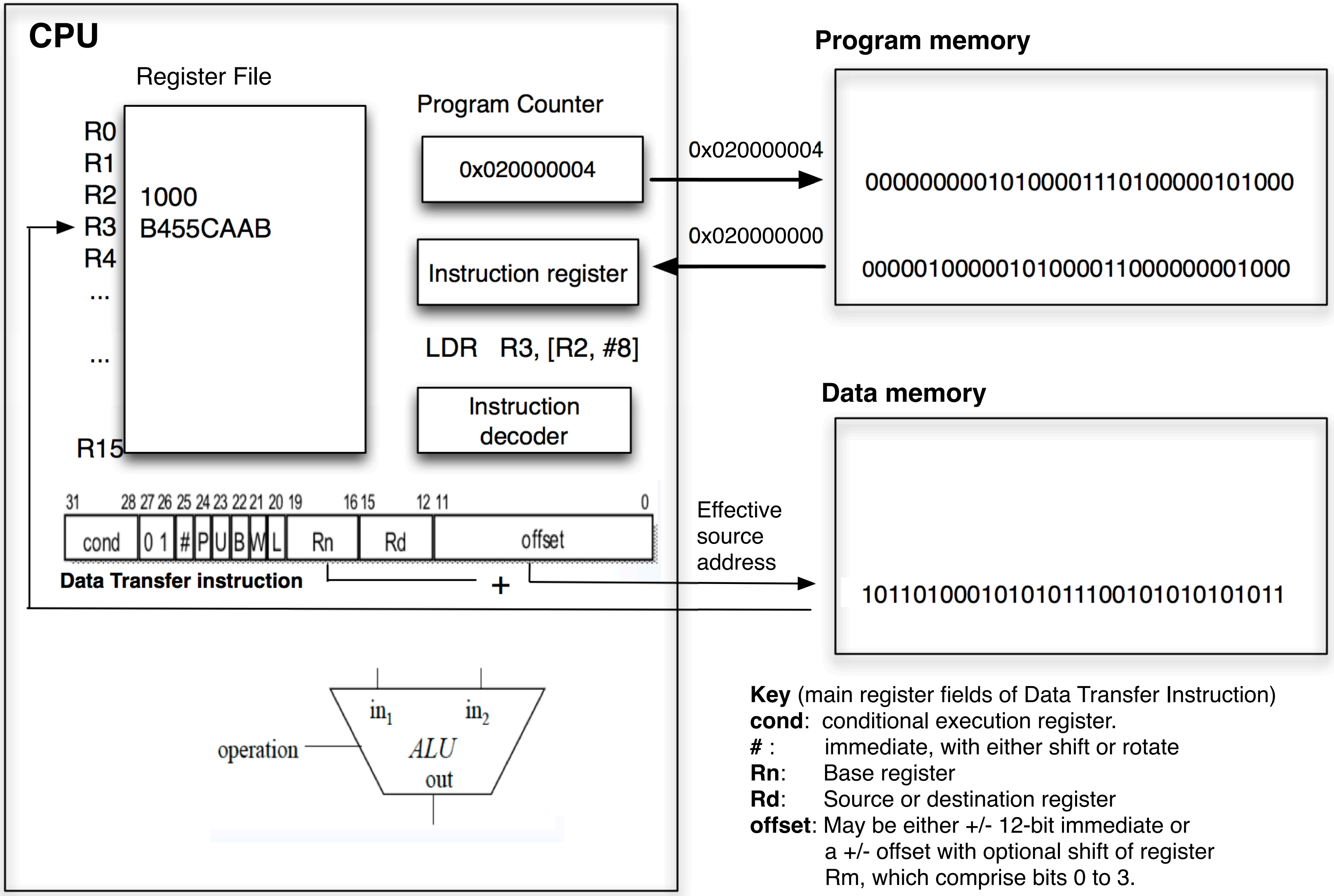
Status bits & the Condition Code Register (CCR)

Current Program Status Register (CPSR) in ARM CPUs

- The condition code register comprises four conditional flags, commonly called *status bits*.
- It provides feedback to the control unit on instruction operations
- CCR comprises of the following bit fields (common to all ISAs):
  - N: Negative: the last ALU operation produced a negative result.
  - Z: Zero: the last ALU operation produced a zero result.
  - C: Carry: the last ALU or shift operation generated a carry out.
  - V: oVerflow: the last ALU operation generated an overflow result.



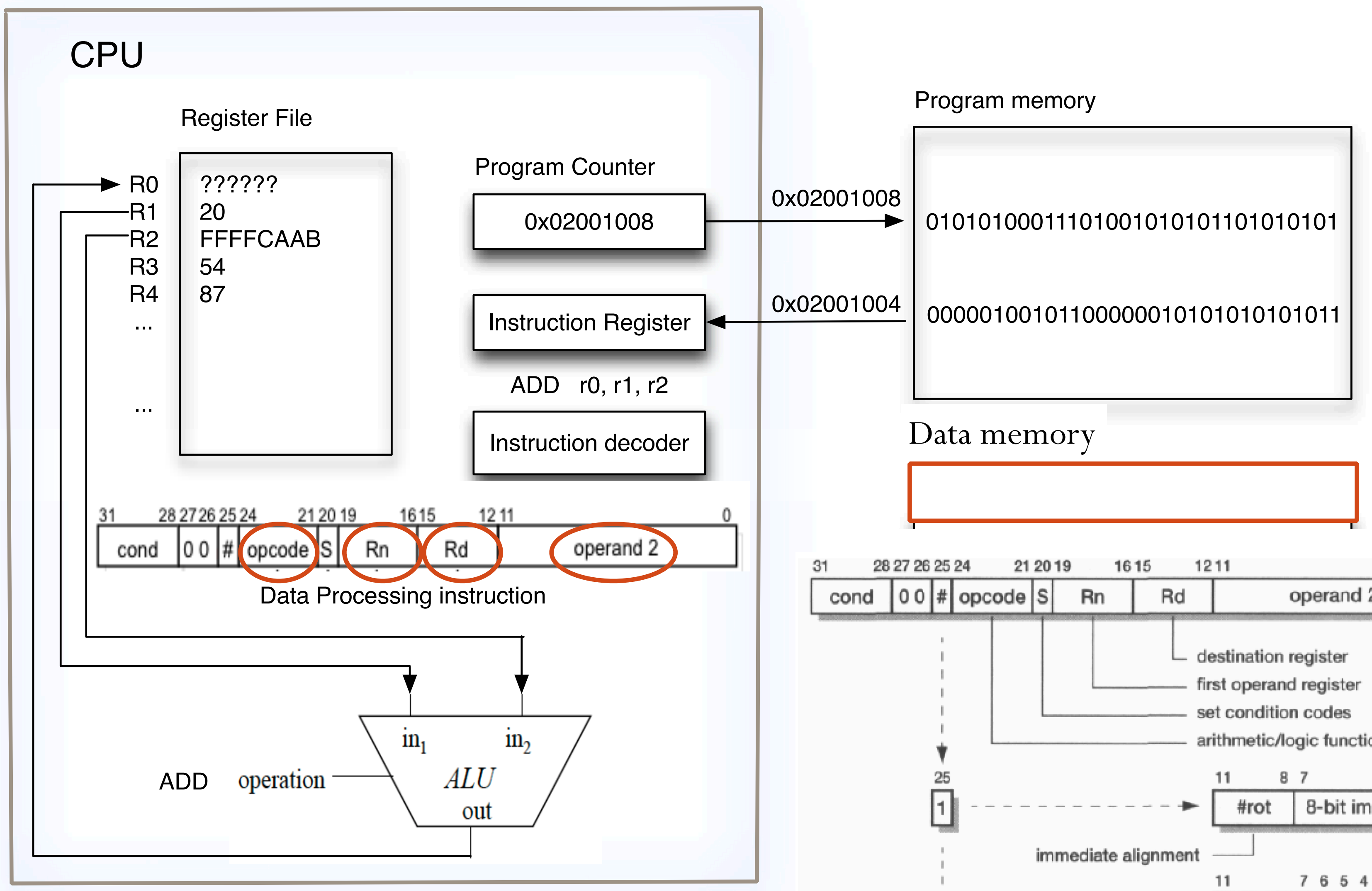
Load data instructions

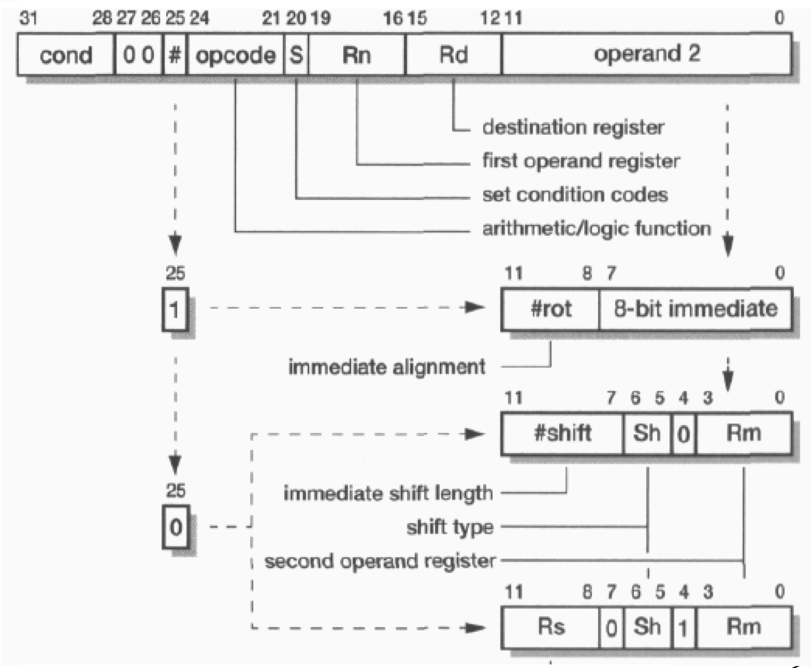


Key (main register fields of Data Transfer Instruction)

cond:	conditional execution register.
# :	immediate, with either shift or rotate
Rn:	Base register
Rd:	Source or destination register
offset:	May be either +/- 12-bit immediate or a +/- offset with optional shift of register Rm, which comprise bits 0 to 3.

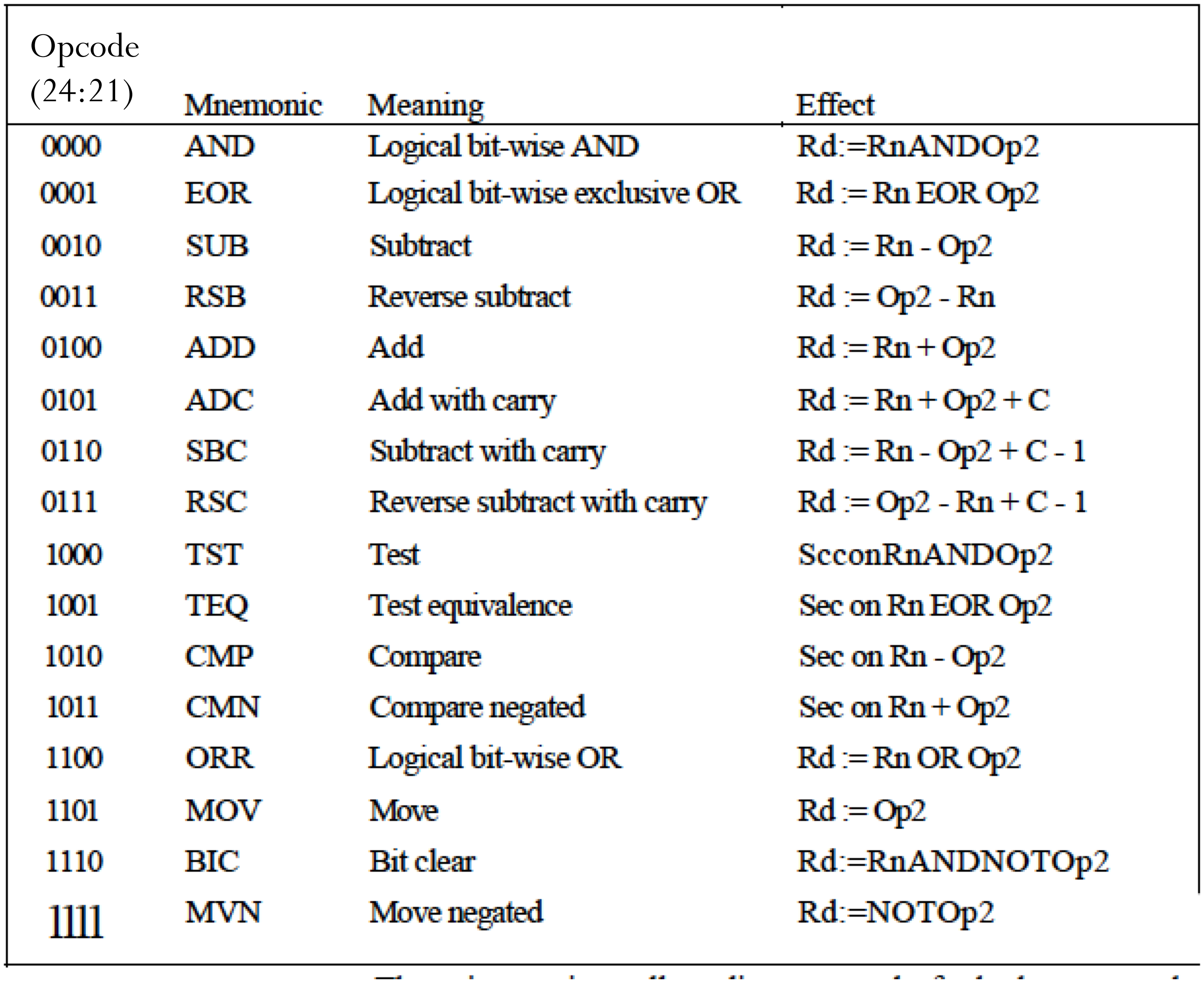
The ARM ISA a distinct subset of load store instructions that are separate from arithmetic instructions. This improves overall efficiency.





Data processing instructions

Data Processing Operations



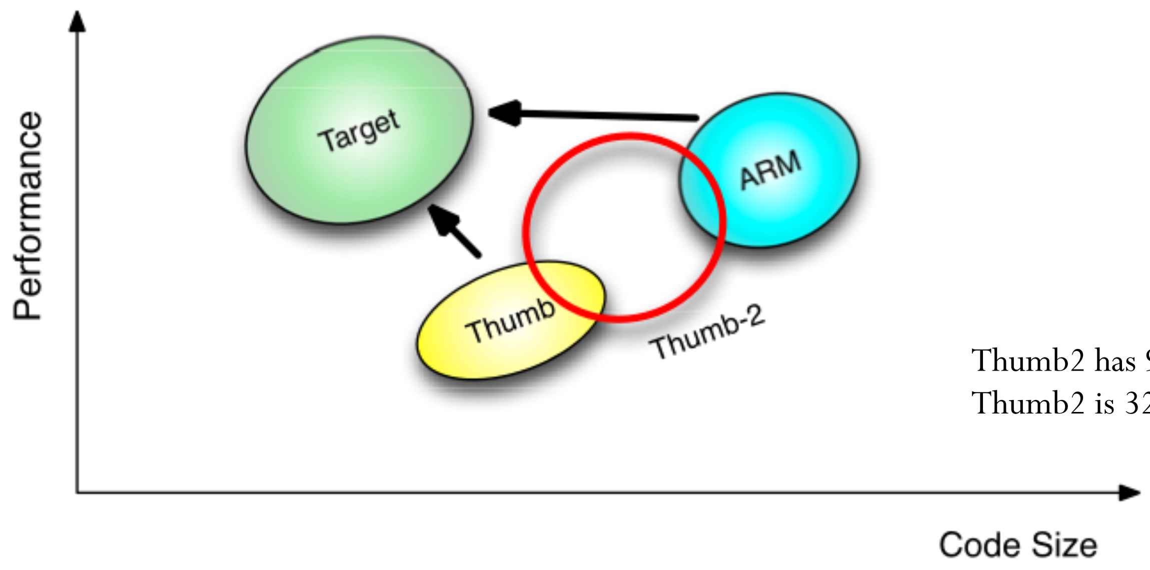
Opcode (24:21)	Mnemonic	Meaning	Effect
0000	AND	Logical bit-wise AND	Rd:=RnANDOp2
0001	EOR	Logical bit-wise exclusive OR	Rd := Rn EOR Op2
0010	SUB	Subtract	Rd := Rn - Op2
0011	RSB	Reverse subtract	Rd := Op2 - Rn
0100	ADD	Add	Rd := Rn + Op2
0101	ADC	Add with carry	Rd := Rn + Op2 + C
0110	SBC	Subtract with carry	Rd := Rn - Op2 + C - 1
0111	RSC	Reverse subtract with carry	Rd := Op2 - Rn + C - 1
1000	TST	Test	ScconRnANDOp2
1001	TEQ	Test equivalence	Sec on Rn EOR Op2
1010	CMP	Compare	Sec on Rn - Op2
1011	CMN	Compare negated	Sec on Rn + Op2
1100	ORR	Logical bit-wise OR	Rd := Rn OR Op2
1101	MOV	Move	Rd := Op2
1110	BIC	Bit clear	Rd:=RnANDNOTOp2
1111	MVN	Move negated	Rd:=NOTOp2

ARM – Core Features

- Load-Store architecture (separate instructions are used to load and store operands in registers).
- All instructions are a fixed length (32-bit).
- Three-address instruction formats are used - two source and one destination operand register.
- All instructions provide conditional execution.
- Most instructions are performed in a single clock cycle.
- Depending on the version, a co-processor interface is offered.
- Thumb (16-bit instruction set) built into the ARM instruction set architecture (ISA).

ARM Cortex M Series ISAs

Features	ARM7TDMI	Cortex-M3
Architecture	ARMv4T (von Neumann)	ARMv7-M (Harvard)
ISA	Thumb / ARM	Thumb / Thumb-2
Interrupts	FIQ / IRQ	NMI + 1-240 specialised
Interrupt Latency	24-42 cycles	12 cycles
Power Consumption	0.28mW/MHz	0.19mW/MHz



Thumb2 has 98% ARM performance
Thumb2 is 32% smaller than ARM

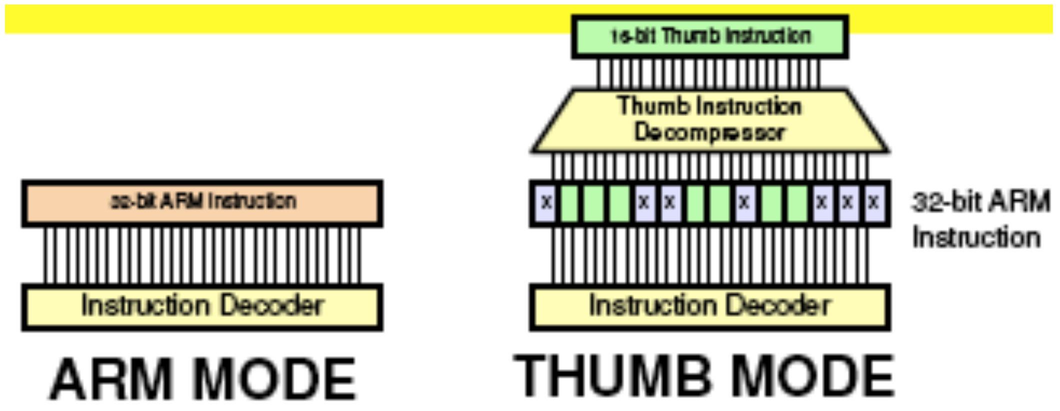
In terms of performance, the Cortex M3 vs M4 are very similar:
https://en.wikipedia.org/wiki/ARM_Cortex-M

C Compiler Register Allocations [4]

Register	Synonym	Special	Role in the procedure call standard
r15		PC	The Program Counter.
r14		LR	The Link Register.
r13		SP	The Stack Pointer.
r12		IP	The Intra-Procedure-call scratch register.
r11	v8		Variable-register 8.
r10	v7		Variable-register 7. Local Variables
r9		v6 SB TR	Platform register. The meaning of this register is defined by the platform standard.
r8	v5		Variable-register 5.
r7	v4		Variable register 4.
r6	v3		Variable register 3. Local Variables
r5	v2		Variable register 2.
r4	v1		Variable register 1.
r3	a4		Argument / scratch register 4.
r2	a3		Argument / scratch register 3. Parameter Passing
r1	a2		Argument / result / scratch register 2.
r0	a1		Argument / result / scratch register 1. Return Result

A more compact 16-bit ISA for the ARM7-TDMI

- ARM instructions are 32-bits wide, however Thumb instructions are 16-bits wide, thus reducing code size; Thumb2 are a combination of 16- and 32-bit.
- ARM7TDMI integrates a real-time instruction de-compressor when executing instructions. Cortex-M3/M4 only supports Thumb and Thumb2 (some only on the M3) instructions.
- Thumb instructions are de-compressed, and expanded into 32-bit instructions. The CPU always executes 32-bit instructions.
- An application can switch between ARM and Thumb modes at any time using the .THUMB or .ARM pseudo-ops (or .THUMB2 for the CM-3 and CM-4) [4].



References

[1] Furber, S., *ARM system-on-chip architecture*, 2nd Ed., Addison-Wesley, 2000.

[2] Atmel Corporation, *AT91 ARM Thumb-based Microcontrollers Datasheet*, Preliminary, November, 2006.

[3] M.M. Mano, & C.R. Kime. *Logic and Computer Design Fundamentals*, 2nd Ed., Prentice Hall, 2001.

Short-answer exercises

1. The condition code register defined in Slide-8 must reside somewhere within the CPU. For most ISAs, is it located in the datapath or control unit?
2. What do all ISAs have in common?
3. Write a instruction operation in register transfer language that subtracts the contents of register r0 from register r1, with a borrow, and places the results in register r6.
4. Considering the four general instruction set architecture types outlined in Slide-6, which general ISA (out of the 4) is used in the example shown in Slide-10, and why?
5. Slide-10 shows an example data processing instruction but data memory is not used. Why not?
6. In Slide-7, Item-4 gives two examples of instruction types that execute in stages. How can a 3-stage instruction, or worse, a 5-stage instruction, execute in one CPU cycle?