09 ARM Architecture
CSC 230

Department of Computer Science
University of Victoria

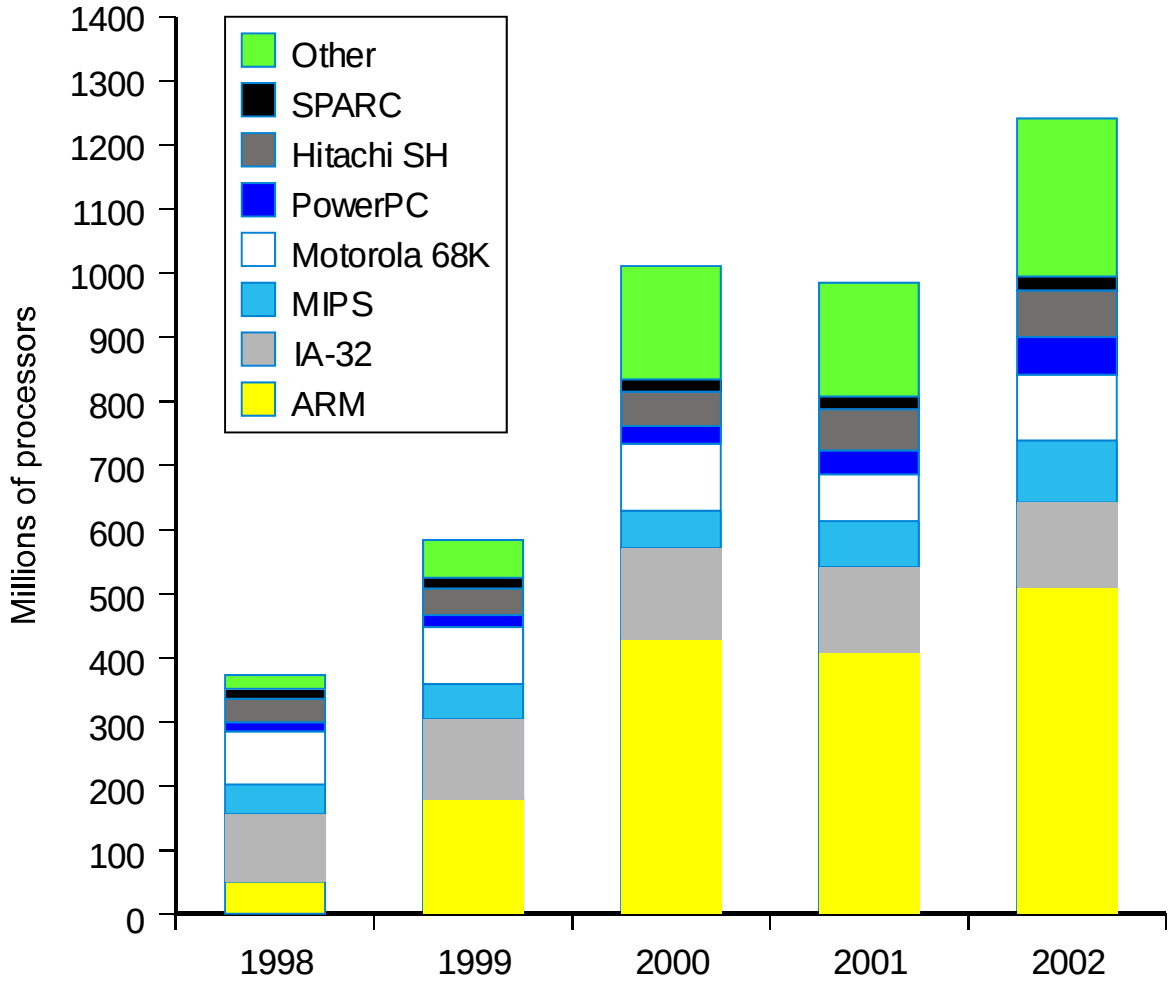
M&H: chapter 4 translated from ARC
ARM Manual
Stallings chapters 12,13 (skip Intel portions)

Background history of the ARM processor

- RISC concepts from Stanford & Berkeley mid 80's
- Acorn Computers (Cambridge, UK) 1983-1985
- Success of Acorn's BBC microcomputer (8-bit processor widely used)
- In 1983 most 16-bit CISC processors were slower than memory!
- Acorn was a small company, no resources to compete commercially with much bigger processor, yet they had clever design ideas
- Berkeley RISC paper showed a RISC design done by a handful of grad students in less than 1 year
- ARM Limited in 1990 → ARM = Advanced/Acorn RISC Machines
- Market leader for low-power and cost-sensitive embedded systems
- ARM supported by strong development tools

Why ARM?

- ARM is one of the most licensed and thus widespread processor cores in the world
- Used especially in portable devices due to low power consumption and reasonable performance (MIPS / watt)
- Several interesting extensions available, for example Thumb instruction set and Jazelle Java machine


Millions of processors
1400
1300
1200
1100
1000
900
800
700
600
500
400
300
200
100
0
Other
SPARC
Hitachi SH
PowerPC
Motorola 68K
MIPS
IA-32
ARM
1998
1999
2000
2001
2002

From a Newsletter on Investment, May 2013

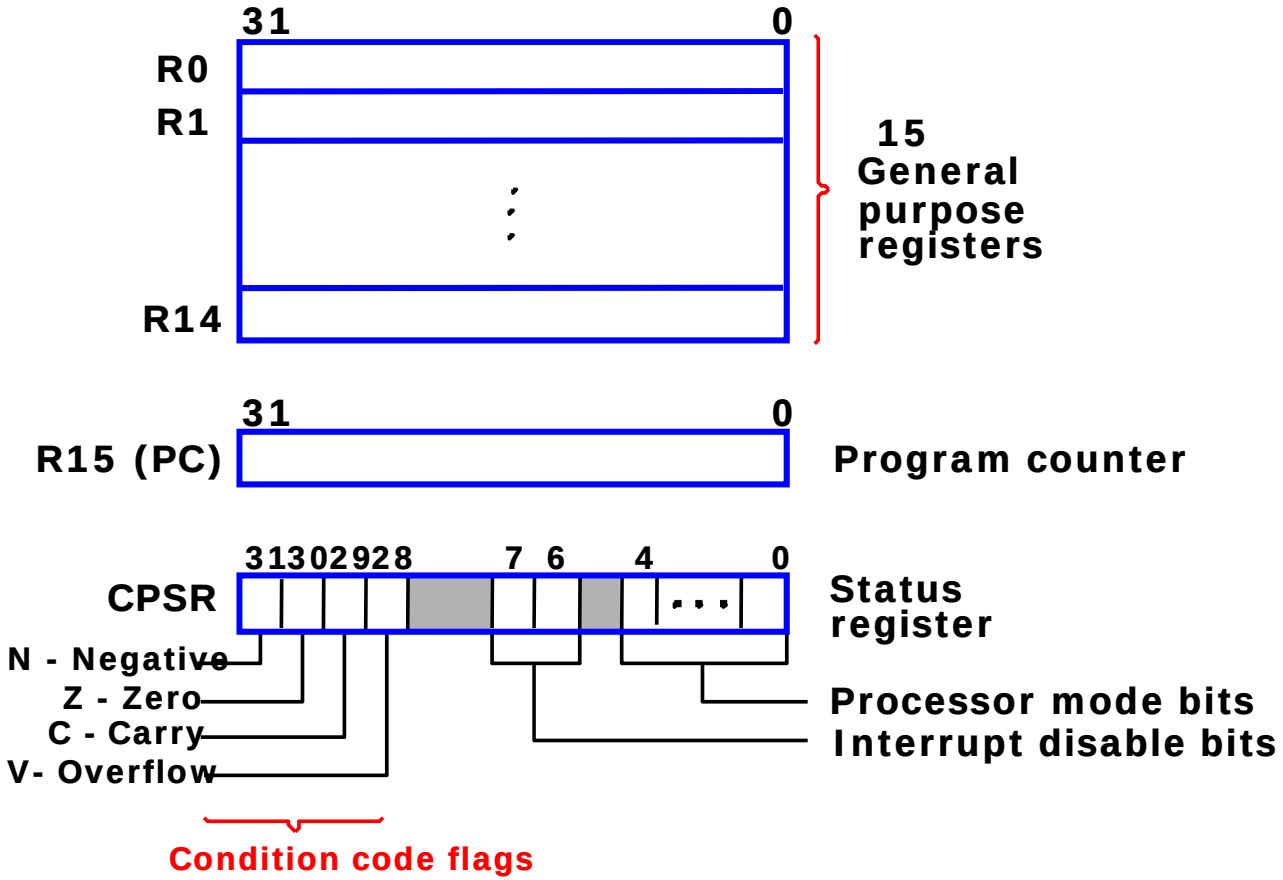
Instead of manufacturing its own CPUs, ARM develops processor and other technology designs that it licenses to other companies for a one-off licence fee while also receiving royalty payments for every ARM-based chip sold by the licencee.

ARM's chips go into a wide range of mobile, consumer and embedded products such as mobile handsets, tablets, laptops, digital cameras and modems. Due to its low-power and battery-saving features, it is the chip of choice for a majority of mobile devices. Its designs are used by companies like **Apple**, **NVIDIA**, **Microsoft**, and **Qualcomm**.

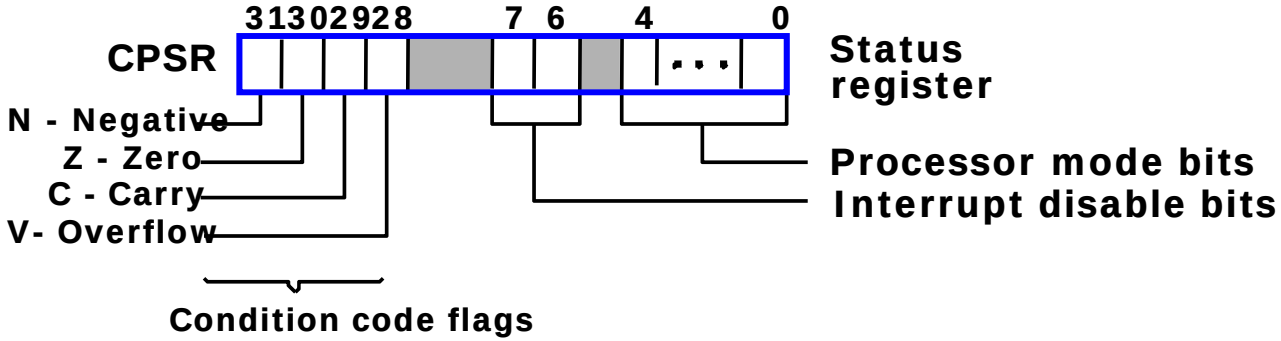
ARM has 95% of the market share in mobile phones, 35% in networking, 45% in digital TVs, and 18% in microcontrollers.

Last year, the company sold around 9 billion ARM-based processor chips, accounting for 32% of the total market. Its closest rival, **Intel**, had 16%

Arm Register Structure



Current Program Status Register (CPSR)



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 operation produced a carry-out (either from arithmetic or from shifting)

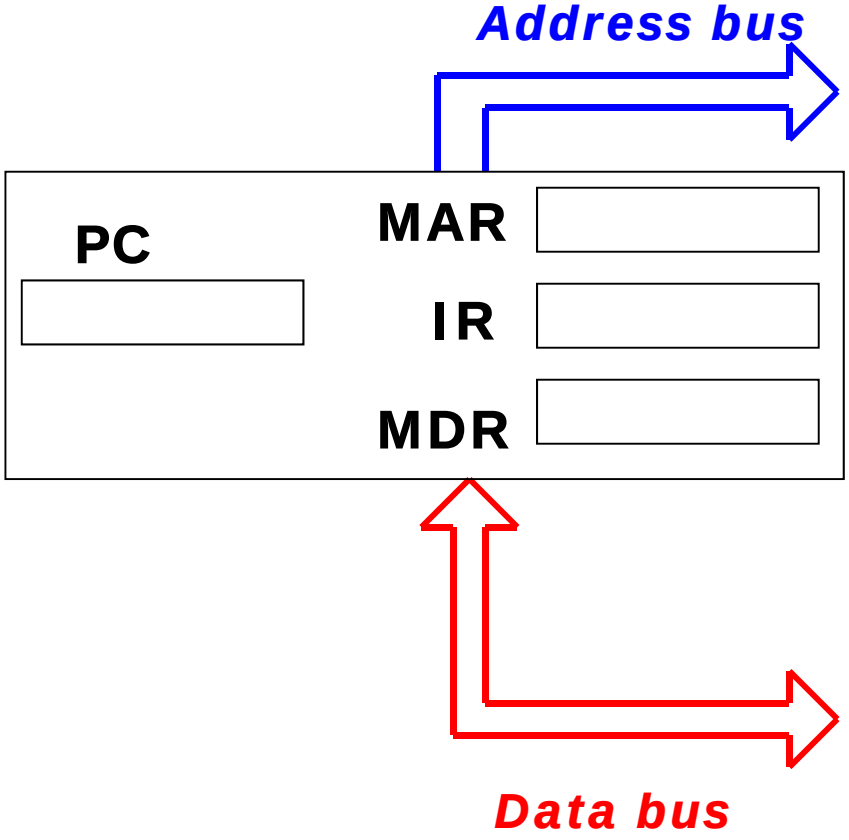
V: oVerflow - the last ALU operation produced an overflow in the sign bit

➔ *most bits are protected in user mode*

R0	R1	...	R13 = SP	R14 = LR	R15 = PC
00000000	00000000		00000000	00000000	00000000

CPSR

00000000



memory addresses	memory content
0xC000	028F101C
0xC004	06112000
0xC008	028F3018
0xC00C	06134000
0xC010	00825004
0xC014	028F6010
0xC018	02165000
0xC01C	00000005
0xC020	00000006
0xC024	00000000

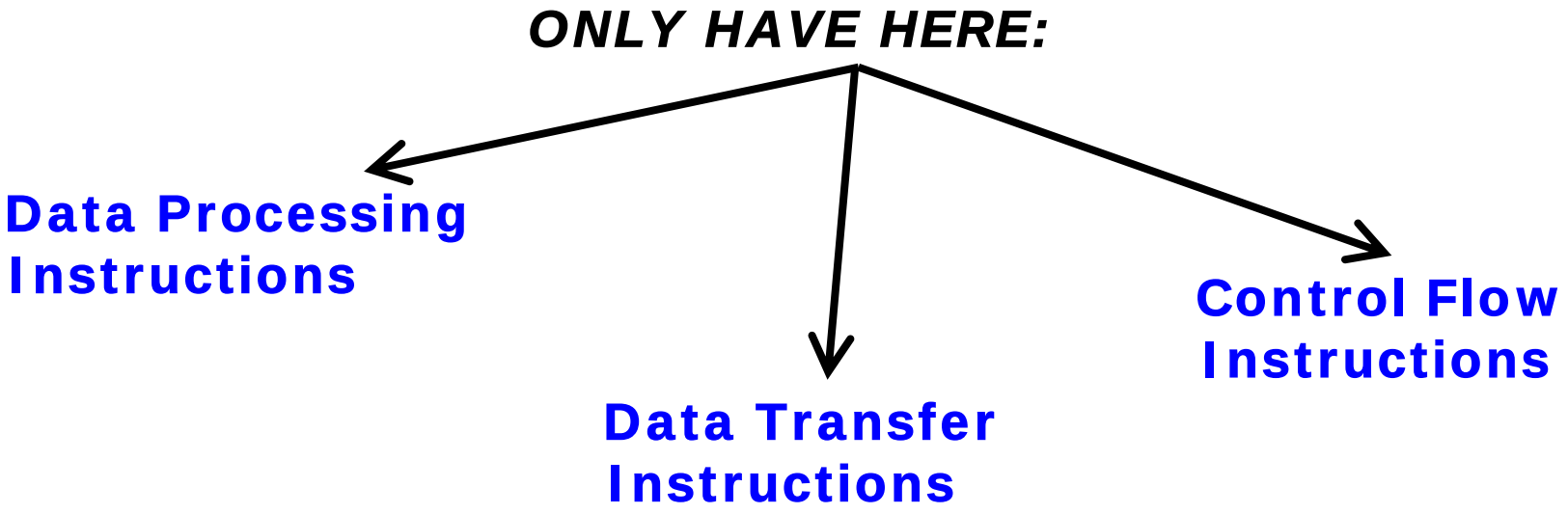
Load-store architecture

- Typical of RISC processors
- ***Instruction set only processes values in registers and places results in registers***
- The ***only*** operations applied to memory states are:

  Load ➔ copy FROM a memory location INTO a register

  Store ➔ copy FROM a register INTO a memory location

- In CISC usually it is allowed to have computation where one operand is a register and another is in memory



ARM Instruction Set

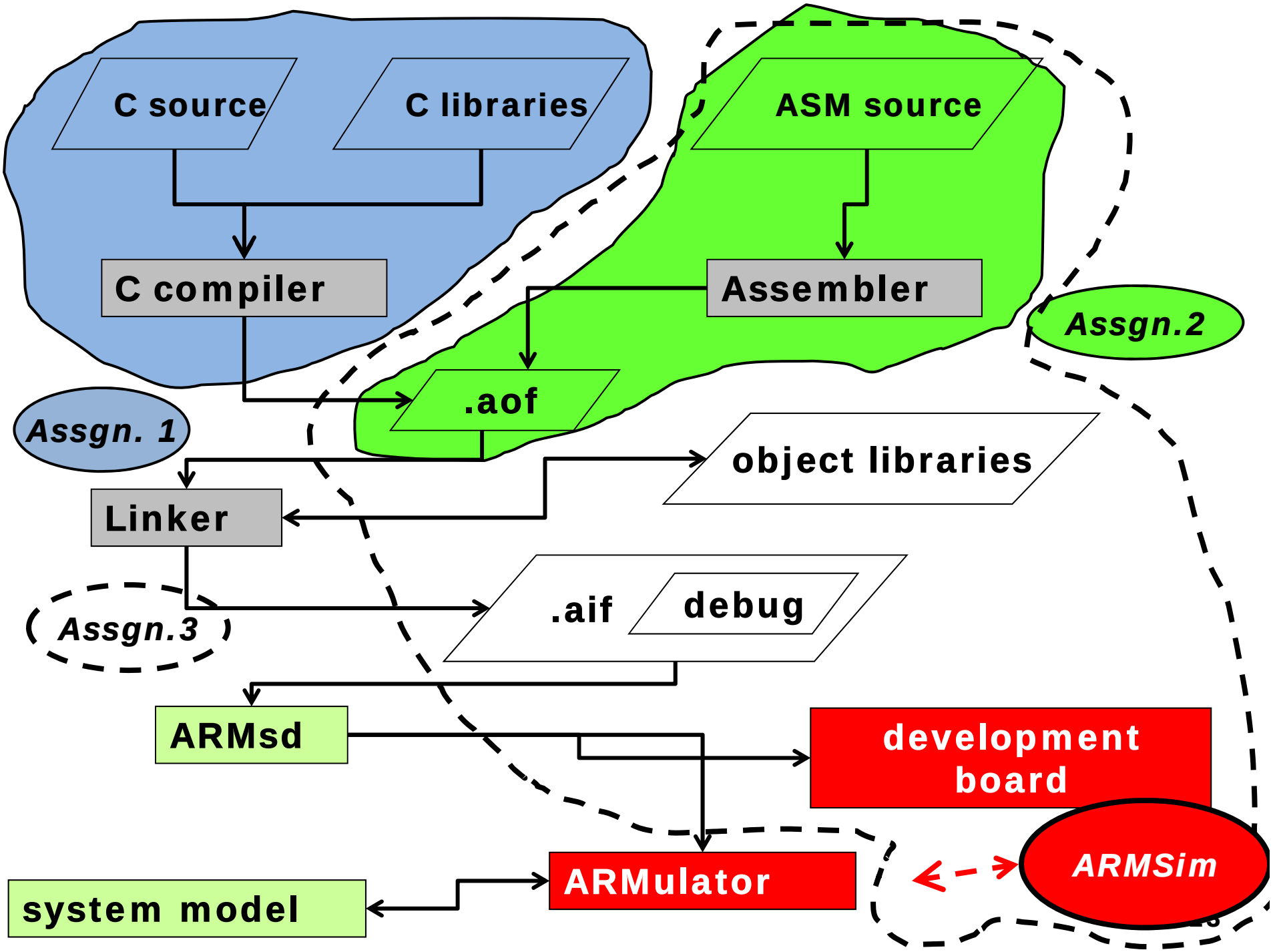
- all instructions are 32-bit wide, aligned on 4-byte boundaries
- load-store architecture
- 3-operands data processing instructions
- very powerful load and store multiple register instructions
- ability to perform a general shift operation and a general ALU operation in a single instruction executing in a single clock cycle

Of interest later in the course:

- *I/O peripherals are handled as memory-mapped devices with interrupt support*
- *internal registers to I/O devices appear as addressable locations and use same load-store instructions*
- *support of a range of interrupts*
- *conditional execution of every instruction (ignore for this course)*

ARM Development Tools

- ARM is widely used as an embedded controller
- Tools are intended for cross-development
  - they can run on a different architecture from the one for which they produce code
  - from PC running Windows to Unix
- sold as “IP cores”
  - Examples:
    - Custom A4 (Iphone, Apple TV) is the Apple version of ARM
    - Xscale from Intel is an ARM


C source
C libraries
ASM source
C compiler
Assembler
Assgn.2
.aof
Assgn. 1
object libraries
Linker
.aif
debug
Assgn.3
ARMsd
development board
ARMSim
ARMulator
system model

Instruction Types

- **Data Processing: Arithmetic, Logic**

  only in registers

  ➔ register direct addressing modes or immediate

- **Data movement: Load, Store**

  index addressing modes with autoincrements

- **Control flow: Branching**

  conditions based on CPSR bits

- **Extra in ARM: Conditional execution *(maybe)***

  for ALL instructions

  it avoids some branching and comparing

Instruction Types Examples: part 1

```
ADD   r1,r2,r3
ORR   r1,r2,#3
MOV   r1,r2
MOV   r3,#0xA
MUL   r1,r2,r3
```

operands ONLY in registers

```
LDR   r1,=NUM
STR   r2,[r1],#4
STR   r2,[r1,r3]
LDRB  r3,[r1]
```

load addresses
load/store values
load constants
indexed addressing modes

```
BNE   loop
BL    subroutine
CMP   R1,#3
```

based on condition codes previously set

Table 2.1
Generic addressing modes

Name	Assembler syntax	Addressing function
Immediate	#Value	Operand = Value
Register	R*i*	EA = R*i*
Absolute (Direct)	LOC	EA = LOC
Indirect	(R*i*) (LOC)	EA = [R*i*] EA = [LOC]
Index	X(R*i*)	EA = [R*i*] + X
Basewithindex	(R*i*,R*j*)	EA = [R*i*] + [R*j*]
Basewithindex andoffset	X(R*i*,R*j*)	EA = [R*i*] + [R*j*] + X
Relative	X(PC)	EA = [PC] + X
Auto increment	(R*i*)+	EA = [R*i*] ; Increment R*i*
Auto decrement	–(R*i*)	Decrement R*i* ; EA = [R*i*]

ARM?

General as shown in many books

EA = effective address
Value = a signed number

General Addressing Modes (not ARM)

ADDRESSING MODE	SYNTAX	MEANING
Immediate	#K	K
Direct (Absolute)	K	Memory [K]
Register Direct	Ri	content of Ri
Indirect	(K)	Memory[Memory[K]]
Register Indirect / Based / Indexed	(Rn) or [Rn]	Memory[Rn]
Register Based with offset	(Rn,#K) or [Rn,#K]	Memory[Rn+K]
Register Indexed with offset	(Rn,Rj) or [Rn,Rj]	Memory[Rn+Rj]
Register Based and Indexed	(Rn,Rj,#K) or [Rn,Rj,#K]	Memory[Rn+Rj+K]