Systems Architecture

The ARM Processor

The ARM Processor

- ARM: *Advanced RISC Machine*
  First developed in 1983 by Acorn Computers
  ARM Ltd was formed in 1988 to continue development
- Advantages of the ARM

RISC:	Reduced Instruction Set Computer
Low power:	good for mobile computing and battery operated devices
Licensed:	Developers can extend the chip in any way they require
Sales:	Outsold all other processors in the last three years

Assembler Programming

- CPU executes binary *machine code* (aka *object code*)
- We write *assembly code* (human readable-*ish*)
- Format of assembly code is CPU dependent
- An *assembler* converts assembly code into machine code
- A *compiler* converts a high-level programming language into assembler which is then assembled into machine code

Processor modes

The ARM has seven processor modes

Processor mode		Description
User	usr	Normal program execution mode
FIQ	fiq	Fast Interrupt for high-speed data transfer
IRQ	irq	Used for general-purpose interrupt handling
Supervisor	svc	A protected mode for the operating system
Abort	abt	Virtual memory / memory protection
Undefined	und	Undefined Instructions Provides support for developer extensions
System	sys	Runs privileged operating system tasks

All ARM Registers

	Privileged Modes					
		Exception Modes				
usr	sys	svc	abt	und	irq	fiq
R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	R8
R9	R9	R9	R9	R9	R9	R9
R10	R10	R10	R10	R10	R10	R10
R11	R11	R11	R11	R11	R11	R11
R12	R12	R12	R12	R12	R12	R12
R13	R13	R13	R13	R13	R13	R13
R14	R14	R14	R14	R14	R14	R14
PC	PC	PC	PC	PC	PC	PC
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR	SPSR	SPSR	SPSR	SPSR	SPSR

ARM Registers

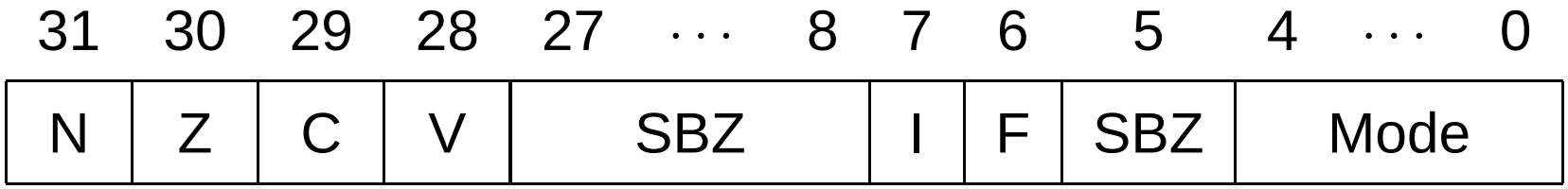
- R0 – R7
  General-purpose unbanked registers
  Same register for *all* modes
  Used for Parameter Passing
- R8 – R12
  General-purpose banked registers
  All modes share the same register except
  Fast Interrupt (fiq) mode which has its own
  Used as *local* registers
- R13 – R14
  Each mode has it's own register:
  R13 – The Stack Pointer (aka *SP*)
  R14 – The Link Register (aka *LR*)
- R15 (aka *PC*)
  All modes share the same program counter

General-Purpose Registers

- 13 General Purpose Registers (R0 – R12)
  Use as (32-bit) variables
  Fast access as register are kept on-chip
  Relies on programmers memory
- Multi-Length Access
  Load and Store instructions can access just the lower 8-bits (Byte) or 16-bits (Halfword) of a 32-bit register
- Signed/Unsigned Access
  When accessing Bytes or Halfwords, what happens to the upper 24- or 16-bits:
  Unsigned: Top bits are set to zero
  Signed: Top bits are set to preserves the sign

Process Status Register

CPSR Current Status Shared by all modes
SPSR Saved Status Each supervisor modes has own



N	True if result of last operation is Negative
Z	True if result of last operation was Zero or equal
C	True if an unsigned borrow (Carry over) occurred Value of last bit shifted
V	True if a signed borrow (oVerflow) occurred
I	True if IRQ interrupts are disabled
F	True if FIQ (Fast) interrupts are disabled
Mode	Processor mode: usr, sys, svc, abt, und, IRQ, or FIQ
SBZ	Should Be Zero — bits are Unused

Exceptions

- Exceptions can be caused by internal and external events

- When an exception occurs:
  Current Processor Status is preserved
  Program execution is stopped
  Processor mode is changed
  Processor executes an *event handler*

- At the end of the *event handler*:
  Processor Status is restored
  Processor mode is restored
  Execution returns to user program

Possible Exceptions

Mode	*Exception / Description*
svc	**Reset** On power up or system reset
und	**Undefined** Attempt to execute an undefined instruction allows for extend instruction set
svc	**Software Interrupt** (SWI) Allows user programs to call the operating system
abt	**Prefetch Abort** Attempt to execute an invalid instruction
abt	**Data Abort** Attempt to access non-aligned memory
IRQ	**Interrupt Request** External device requesting attention
FIQ	**Fast Interrupt Request** Same as IRQ but for impatient devices

Instructions and Addresses

- All instructions require a *destination* and at least one *source* which is given in terms of an *effective address*
- Data Processing effective addresses ($\langle op1 \rangle$):

Immediate	$\#nnn$	Scaled Immediate	Rn, $\langle$*shift*$\rangle$ $\#nnn$
Register	Rn	Scaled Register	Rn, $\langle$*shift*$\rangle$ Rs

- Memory Access effective addresses ($\langle op2 \rangle$):

	Immediate	Register	Scaled Register
Offset	[Rn, $\#nnn$]	[Rn, Rm]	[Rn, Rm, $\langle$*shift*$\rangle$ $\#nnn$]
Pre-indexed	[Rn, $\#nnn$]!	[Rn, Rm]!	[Rn, Rm, $\langle$*shift*$\rangle$ $\#nnn$]!
Post-Indexed	[Rn], $\#nnn$	[Rn], Rm	[Rn], Rm, $\langle$*shift*$\rangle$ $\#nnn$

- Where $\langle$*shift*$\rangle$ is one of:

LSL	Logical Shift Left	ROR	Rotate Right
LSR	Logical Shift Right	RRX	Rotate Right eXtended
ASR	Arithmetic Shift Right		

Assembler Code Terminology

Label	*Mnemonic* or *Directive*	*Operands*	*Comment*
`Main`	`MOV`	`r0, #0`	`; move 0 into R0`

label	Give a name to the location of the instruction
mnemonic	Human readable name given to an instruction `MOV` (Move) or `LDR` (Load Register)
operands	Arguments for a given instruction *effective address* (Data or Memory)
directive	Instructions to the assembler `END` (End of program source)

Example Assembly Program

```
1.  * === Example Program ===
2.
3.          AREA    Program, CODE, READONLY
4.          ENTRY
5.
6.  Main    MOV     r0, #0              ; R0 ← 0
7.
8.  Repeat  LDRB    r1, [r12, #0]       ; R1(7:0) ← Input
9.          CMP     r1, #0              ; If R1 == 0 then
10.         BEQ     Done                ;    PC ← Done
11.
12.         ADD     r0, r0, r0, LSL #2  ; R0 ← R0 + R0 * 4
13.         ADD     r0, r1, r0, LSL #1  ; R0 ← R1 + R0 * 2
14.         BAL     Repeat              ; PC ← Repeat
15.
16. Done    STR     r0, [r12, #0]       ; Output ← R0
17.         SWI     &11
18.
19.         END
```

Assembler Directives

AREA	Declare Program **Area** Set the type of program area (code or data) Memory type: ReadWrite or ReadOnly
ENTRY	Declare program's **entry** point (address of the program's first instruction)
EQU	**Equate** label with a value Associate a name with a value
END	**End** of source code
DCB	**Define Constant Byte** (8-bits)
DCW	**Define Constant Word** (16-bits)
DCD	**Define Constant Data** (32-bits) Value placed in memory at program start up.