# ARM Workshop on Blueboard

# Part-1

# Workshop Overview

## Day 1

ARM Arch

Tools setup

Blue board

Programming Intro

ARM Programming:

TIMER/COUNT, PWM

Interrupts, UART

## Day 2

ARM Programming:

UART Int,ADC,RTC,LCD

I2C,SPI, Watchdog Timer,

## AGENDA

- Introduction
- Pipeline
- Registers
- Exception Modes
- Instruction Sets

# What is ARM?

## INTRODUCTION

- The ARM is a 32-bit reduced instruction set computer (RISC) instruction set architecture (ISA) developed by ARM Holdings.

- ARM also known as Advance RISC Machine

# Where is ARM?

# ARM Core Family

| Application Cores | Embedded Cores | Secure Cores |
|---|---|---|
| ARM720T | ARM7EJ-S | SecureCore SC100 |
| ARM920T | ARM7TDMI | SecureCore SC110 |
| ARM922T | ARM7TDMI-S | SecurCore SC200 |
| ARM926EJ-S | ARM946E-S | SecurCore SC210 |
| ARM1020E | ARM966E-S | |
| ARM1022 | ARM968E-S | |
| ARM1026EJ-S | ARM996HS | |
| ARM11 MPCore | ARM1026EJ-S | |
| ARM1136J(F)-S | ARM1156T2(F)-S | |
| ARM1176JZ(F)-S | ARM Cortex-M0 | |
| ARM Cortex-A8 | ARM Cortex-M1 | |
| ARM Cortex-A9 | ARM Cortex-M3 | |

# ARM Core Family

**T: Thumb**
**D: On-chip debug support**
**M: Enhanced multiplier**
**I: Embedded ICE hardware**
**T2: Thumb-2**
**S: Synthesizable code**
**E: Enhanced DSP instruction set**
**J: JAVA support, Jazelle**
**Z: Should be TrustZone?**
**F: Floating point unit**
**H: Handshake, clockless design for synchronous or asynchronous design**

# ARM Core Family

**ARM processor core + cache + MMU = ARM CPU cores**

**ARM6 → ARM7**

- 3-stage pipeline
- Keep its instructions and data in the same memory system
- Thumb 16-bit compressed instruction set
- On-chip Debug support, enabling the processor to halt in response to a debug request
- Enhanced Multiplier, 64-bit result
- Embedded ICE hardware, give on-chip breakpoint and watchpoint support

# ARM Core Family

**ARM8 → ARM9 → ARM10**

**ARM9**

– 5-stage pipeline (130 MHz or 200MHz)

– Using separate instruction and data memory ports

**ARM 10 (1998. Oct.)**

– High performance, 300 MHz

– Multimedia digital consumer applications

– Optional vector floating-point unit

**ARM11 (2002 Q4)**

–8-stage pipeline

– Addresses a broad range of applications in the wireless, consumer, networking and automotive segments

–Support media accelerating extension instructions

–Can achieve 1GHz & Support AXI

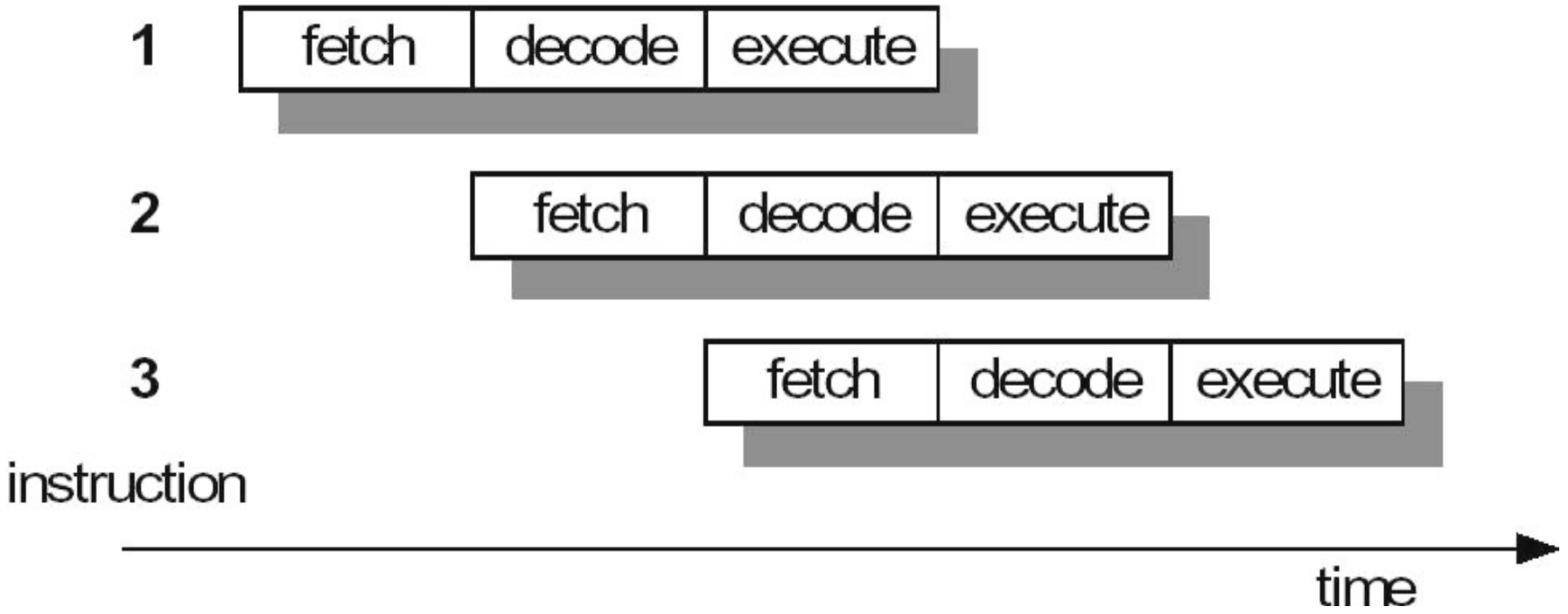
# ARM Cores & Arch Ver

| Core | Architecture |
|---|---|
| ARM1 | v1 |
| ARM2 | v2 |
| ARM2as, ARM3 | v2a |
| ARM6, ARM600, ARM610 | v3 |
| ARM7, ARM700, ARM710 | v3 |
| ARM7TDMI, ARM710T, ARM720T, ARM740T | v4T |
| StrongARM, ARM8, ARM810 | v4 |
| ARM9TDMI, ARM920T, ARM940T | V4T |
| ARM9E-S, ARM10TDMI, ARM1020E | v5TE |
| ARM10TDMI, ARM1020E | v5TE |
| ARM11 MPCore, ARM1136J(F)-S, ARM1176JZ(F)-S | v6 |
| Cortex-A/R/M | v7 |

# The Pipeline

# Three Stage Pipeline

The pipeline is used to overcome the delay caused by instruction fetching and decoding before execution.



# Three Stage Pipeline

**Fetch**

– The instruction is fetched from memory and placed in the instruction pipeline

**Decode**

– The instruction is decoded and the data path control signals prepared for the next cycle

**Execute**

– The register bank is read, an operand shifted, the ALU result generated and written back into destination register

**The three stage pipeline has hardware independent stages that execute one instruction while decoding a second and fetching a third.**

PC runs 8 bytes ahead of current execution instruction since it holds the address of the fetching instruction but not the current execution instruction.

*0x4000 LDR PC,[PC,#4] results PC => 0x400C not 0x4004*

# Processor Modes

- **User** : Normal program execution state
- **FIQ** : Data transfer state (fast irq, DMA-type transfer)
- **IRQ** : Used for general interrupt services
- **Supervisor** : Protected mode for operating system support
- **Abort** : Selected when data or instruction fetch is aborted
- **Undef** : Selected when undefined instruction is fetched
- **System** : Operating system ‘privilege’-mode for user

# Registers

# Registers

**ARM has 37 registers all of which are 32-bits long.**

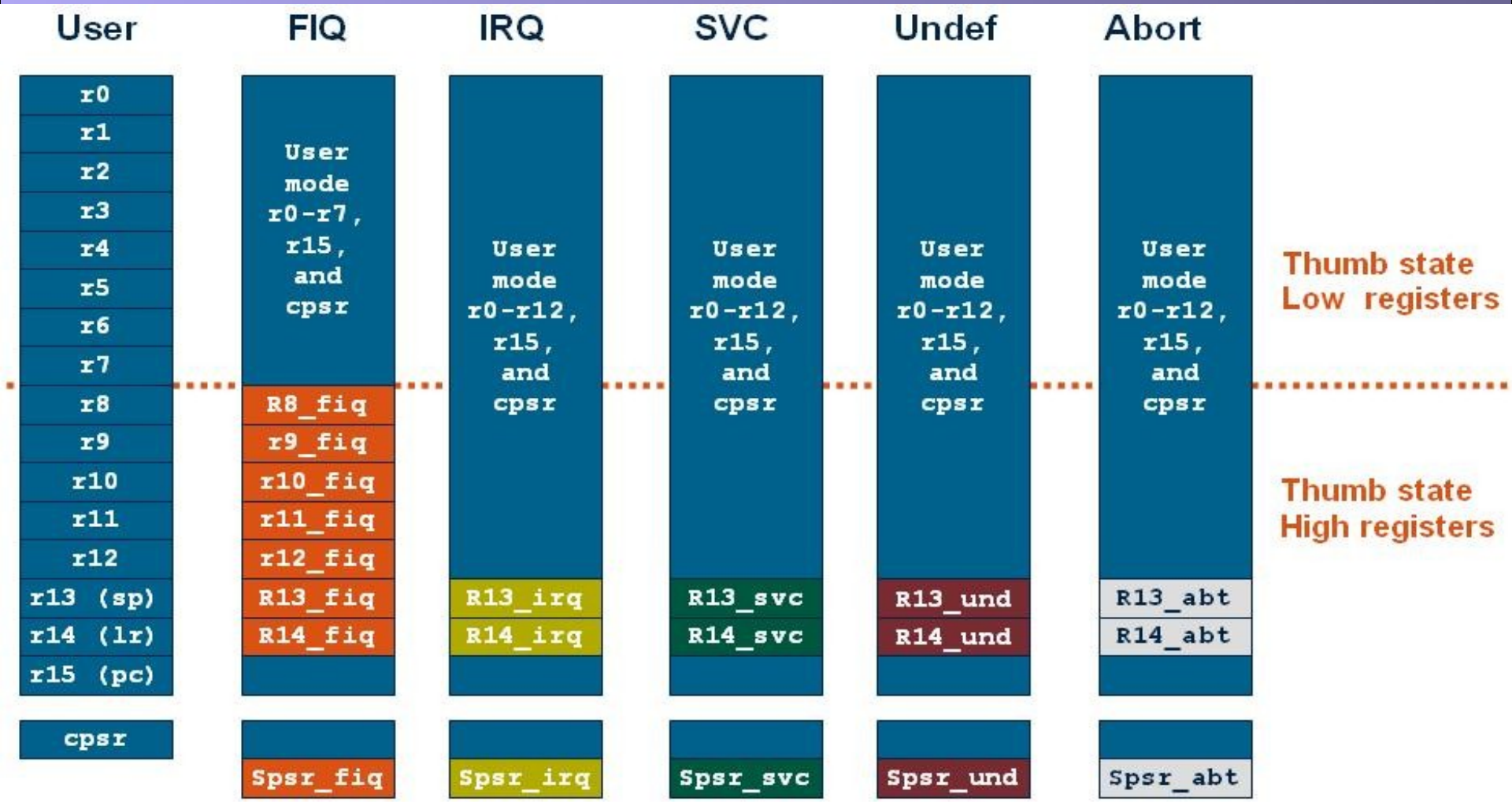
-1 dedicated program counter

-1 dedicated current program status register

-5 dedicated saved program status registers

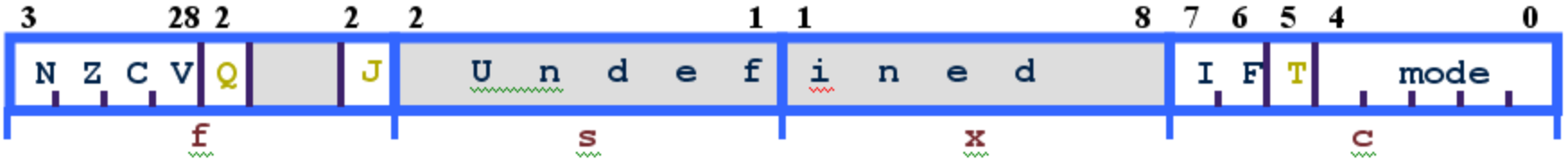
-30 general purpose registers

# Registers




Note: System mode uses the User mode register set

# Registers




**Condition code flags**

N = Negative result from ALU
Z = Zero result from ALU
C = ALU operation Carried out
V = ALU operation Overflowed

**Sticky Overflow flag - Q flag**

Architecture 5TE/J only
Indicates if saturation has occurred

**J bit**

Architecture 5TEJ only
J = 1: Processor in Jazelle state

**Interrupt Disable bits.**

I = 1: Disables the IRQ.
F = 1: Disables the FIQ.

**T Bit**

Architecture xT only
T = 0: Processor in ARM state
T = 1: Processor in Thumb state

**Mode bits**

Specify the processor mode

# Exception Handling

# Exception Handling

**When an exception occurs, the ARM:**

- Copies CPSR into SPSR_<mode>
- Sets appropriate CPSR bits
- Change to ARM state
- Change to exception mode
- Disable interrupts (if appropriate)
- Stores the return address in LR_<mode>
- Sets PC to vector address

**To return, exception handler needs to:**

- Restore CPSR from SPSR_<mode>
- Restore PC from LR_<mode>

# Exception Handling

| Exception | Mode | Address |
|---|---|---|
| Reset | Supervisor | 0x00000000 |
| Undefined instruction | Undefined | 0x00000004 |
| Software interrupt (SWI) | Supervisor | 0x00000008 |
| Prefetch Abort (instruction fetch memory abort) | Abort | 0x0000000C |
| Data Abort (data access memory abort) | Abort | 0x00000010 |
| IRQ (interrupt) | IRQ | 0x00000018 |
| FIQ (fast interrupt) | FIQ | 0x0000001C |

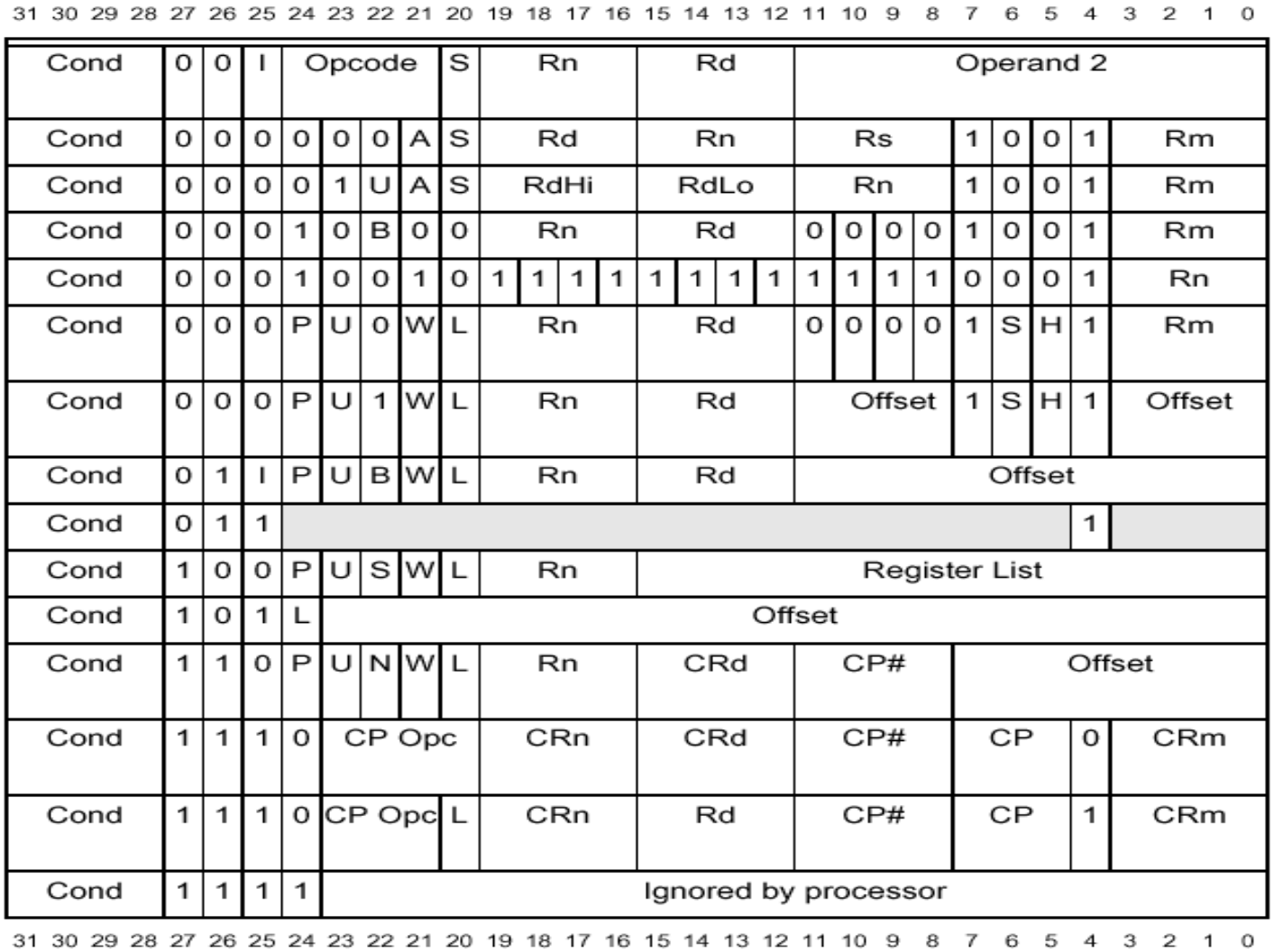
# Instruction Set

# Instruction Set

**ARM's implement two types of instruction sets**

- 32-bit ARM Instruction Set
- 16-bit Thumb Instruction Set

# ARM (32bit) IS




***Data Processing / PSR Transfer***

***Multiply***

***Multiply Long***

***Single Data Swap***

***Branch and Exchange***

***Halfword Data Transfer: register offset***

***Halfword Data Transfer: immediate offset***

***Single Data Transfer***

***Undefined***

***Block Data Transfer***

***Branch***

***Coprocessor Data Transfer***

***Coprocessor Data Operation***

***Coprocessor Register Transfer***

***Software Interrupt***

# ARM (32bit) IS

- Every ARM (32 bit) instruction is conditionally executed.
- The top four bits are ANDed with the CPSR condition codes, If they do not matched the instruction is executed as NOP
- The AL condition is used to execute the instruction irrespective of the value of the condition code flags.
- By default, data processing instructions do not affect the condition code flags but the flags can be optionally set by using "S". Ex: SUBS r1,r1,#1
- Conditional Execution improves code density *and* performance by reducing the number of f

| **Normal** | **Conditional** |
|---|---|
| CMP r3,#0<br>BEQ skip<br>ADD r0,r1,r2<br>skip | CMP r3,#0<br>ADDNE r0,r1,r2 |

# Condition Codes

Each ARM (32bit) Instruction can be prefixed with any of the following conditional code.

| Code | Suffix | Flags | Meaning |
|---|---|---|---|
| 0000 | EQ | Z set | equal |
| 0001 | NE | Z clear | not equal |
| 0010 | CS | C set | unsigned higher or same |
| 0011 | CC | C clear | unsigned lower |
| 0100 | MI | N set | negative |
| 0101 | PL | N clear | positive or zero |
| 0110 | VS | V set | overflow |
| 0111 | VC | V clear | no overflow |
| 1000 | HI | C set and Z clear | unsigned higher |
| 1001 | LS | C clear or Z set | unsigned lower or same |
| 1010 | GE | N equals V | greater or equal |
| 1011 | LT | N not equal to V | less than |
| 1100 | GT | Z clear AND (N equals V) | greater than |
| 1101 | LE | Z set OR (N not equal to V) | less than or equal |
| 1110 | AL | (ignored) | always |

# Condition Codes

**Examples:**

**Set the flags, then use various condition codes**

```
if (a==0) x=0;
if (a>0)  x=1;

CMP     r0,#0
MOVEQ   r1,#0
MOVGT   r1,#1
```

# Branch instructions

**B** Basic branch instruction used to jump forward or backward of up to 32 MB.

**BL** Branch and Link instruction jumps to the destination and stores a return address in R14 (Link Register).

**BX, BLX** Branch, Brach Link and Exchange.
This swaps the instruction sets from ARM to THUMB and vice versa while jumping.

**BXJ** Branch and change to Jazelle state.

# Data Processing Inst.

| | | | | | | |
|---|---|---|---|---|---|---|
| **Arithmetic:** | **ADD** | **ADC** | **SUB** | **SBC** | **RSB** | **RSC** |
| **Logical:** | | **AND** | **ORR** | **EOR** | **BIC** | |
| **Comparisons:** | | **CMP** | **CMN** | **TST** | **TEQ** | |
| **Data movement:** | | **MOV** | **MVN** | | | |

# Multiply Instructions

**MUL, MLA**

## Examples

```
MUL       R1,R2,R3  ; R1:=R2*R3
MLAEQS    R1,R2,R3,R4; Conditionally R1:=R2*R3+R4,
                     ; setting condition codes.
```

**MULL, MLAL**

## Examples

```
UMULL     R1,R4,R2,R3; R4,R1:=R2*R3
UMLALS    R1,R5,R2,R3; R5,R1:=R2*R3+R5,R1 also setting
                     ; condition codes
```

| Mnemonic | Description | Purpose |
|---|---|---|
| **UMULL{cond}{S} RdLo,RdHi,Rm,Rs** | Unsigned Multiply Long | 32 x 32 = 64 |
| **UMLAL{cond}{S} RdLo,RdHi,Rm,Rs** | Unsigned Multiply & Accumulate Long | 32 x 32 + 64 = 64 |
| **SMULL{cond}{S} RdLo,RdHi,Rm,Rs** | Signed Multiply Long | 32 x 32 = 64 |
| **SMLAL{cond}{S} RdLo,RdHi,Rm,Rs** | Signed Multiply & Accumulate Long | 32 x 32 + 64 = 64 |

# Data Transfer Inst.

**Simple Data Transfer Inst.**

| | | |
|---|---|---|
| LDR | STR | Word |
| LDRB | STRB | Byte |
| LDRH | STRH | Halfword |
| LDRSB | | Signed byte load |
| LDRSH | | Signed halfword load |

**Load / Store Multiple Registers**

LDM

STM

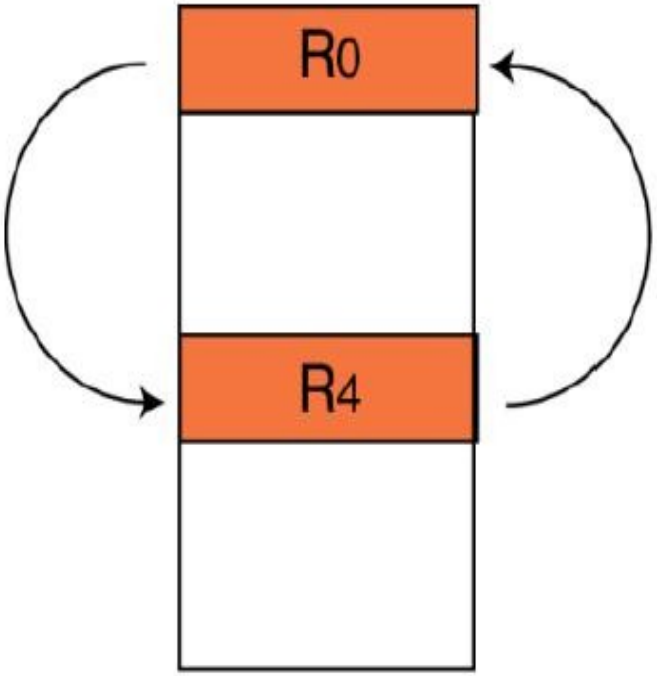
# Swap Instruction

Are also called as semaphore instructions

**SWP R12, R10, [R9]**
; load R12 from address R9 and
; store R10 to address R9

**SWPB R3, R4, [R8]**
; load byte to R3 from address R8 and
; store byte from R4 to address R8

**SWP R1, R1, [R2]**
; Exchange value in R1 and address in R2




The swap instruction allows you to exchange the contents of two registers. This takes two cycles but is treated as a single atomic instruction so the exchange cannot be corrupted by an interrupt.

# Miscellaneous Inst.

## Software Interrupt

Causes an exception trap to the SWI hardware vector

The SWI handler can examine the SWI number to decide what operation has been requested.

By using the SWI mechanism, an operating system can implement a set of privileged operations which applications running in user mode can request.

Ex. SWI #3

## PSR Transfer Instructions

MRS and MSR allow contents of CPSR / SPSR to be transferred to / from a general purpose register.

```
MRS{<cond>} Rd,<psr>                ; Rd = <psr>
MSR{<cond>} <psr[_fields]>,Rm       ; <psr[_fields]> = Rm
```