ARM

The ARM Architecture

THE ARCHITECTURE FOR THE DIGITAL WORLD

ARM

Agenda

- **Introduction to ARM Ltd**
- Programmers Model
- Instruction Set
- System Design
- Development Tools

ARM

ARM Ltd

- **Founded in November 1990**
  - Spun out of Acorn Computers
- **Designs the ARM range of RISC processor cores**
- **Licenses ARM core designs to semiconductor partners who fabricate and sell to their customers.**
  - ARM does not fabricate silicon itself
- **Also develop technologies to assist with the design-in of the ARM architecture**
  - Software tools, boards, debug hardware, application software, bus architectures, peripherals etc

ARM

ARM Partnership Model

ATAP Partners
Tools Partners
ARM in Partnership
RTOS Partners
Software Partners

ARM Powered Products

Intellectual Property

- **ARM provides hard and soft views to licencees**
  - RTL and synthesis flows
  - GDSII layout
- **Licencees have the right to use hard or soft views of the IP**
  - soft views include gate level netlists
  - hard views are DSMs
- **OEMs must use hard views**
  - to protect ARM IP

Agenda

Introduction to ARM Ltd
- **Programmers Model**

Instruction Sets
System Design
Development Tools

Data Sizes and Instruction Sets

- **The ARM is a 32-bit architecture.**
- **When used in relation to the ARM:**
  - **Byte** means 8 bits
  - **Halfword** means 16 bits (two bytes)
  - **Word** means 32 bits (four bytes)
- **Most ARM's implement two instruction sets**
  - 32-bit ARM Instruction Set
  - 16-bit Thumb Instruction Set
- **Jazelle cores can also execute Java bytecode**

The 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

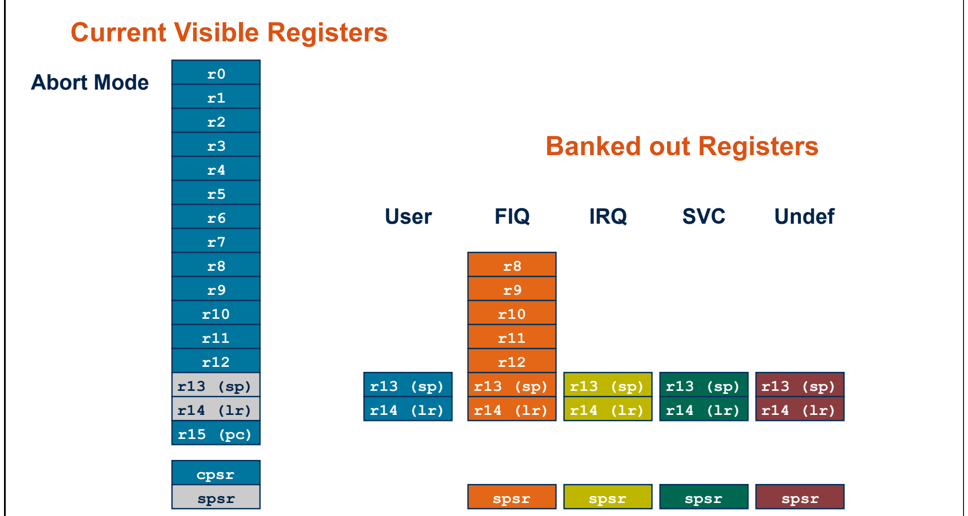
- **The current processor mode governs which of several banks is accessible. Each mode can access**
  - a particular set of r0-r12 registers
  - a particular r13 (the stack pointer, sp) and r14 (the link register, lr)
  - the program counter, r15 (pc)
  - the current program status register, cpsr

Privileged modes (except System) can also access
  - a particular spsr (saved program status register)

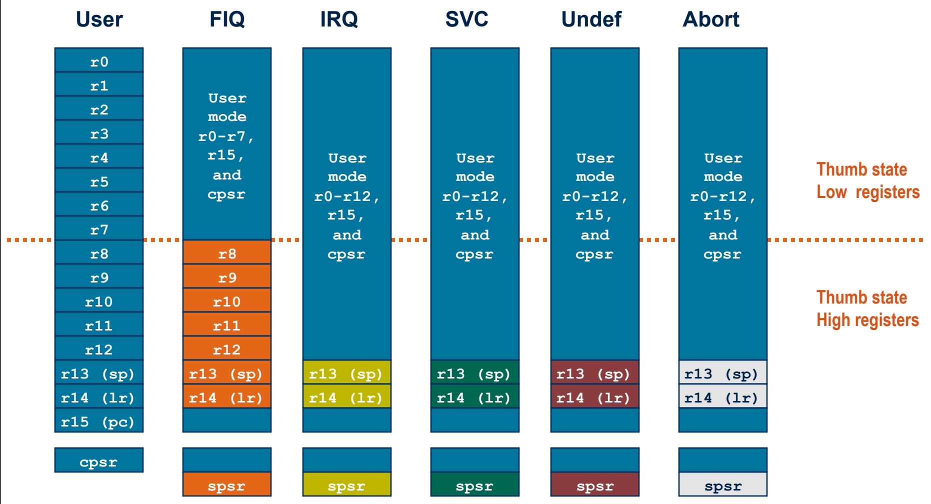
Processor Modes

- **The ARM has seven basic operating modes:**
  - **User** : unprivileged mode under which most tasks run
  - **FIQ** : entered when a high priority (fast) interrupt is raised
  - **IRQ** : entered when a low priority (normal) interrupt is raised
  - **Supervisor** : entered on reset and when a Software Interrupt instruction is executed
  - **Abort** : used to handle memory access violations
  - **Undef** : used to handle undefined instructions
  - **System** : privileged mode using the same registers as user mode

The ARM Register Set

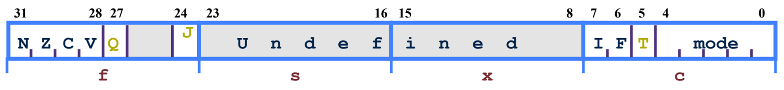


Register Organization Summary


User
FIQ
IRQ
SVC
Undef
Abort
r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13 (sp)
r14 (lr)
r15 (pc)
cpsr
User mode r0-r7, r15, and cpsr
User mode r0-r12, r15, and cpsr
spsr
Thumb state Low registers
Thumb state High registers

Note: System mode uses the User mode register set

Program Status Registers



- **Condition code flags**
  - N = **N**egative result from ALU
  - Z = **Z**ero result from ALU
  - C = ALU operation **C**arried out
  - V = ALU operation o**V**erflowed
- **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

Program Counter (r15)

- **When the processor is executing in ARM state:**
  - All instructions are 32 bits wide
  - All instructions must be word aligned
  - Therefore the **pc** value is stored in bits [31:2] with bits [1:0] undefined (as instruction cannot be halfword or byte aligned).
- **When the processor is executing in Thumb state:**
  - All instructions are 16 bits wide
  - All instructions must be halfword aligned
  - Therefore the **pc** value is stored in bits [31:1] with bit [0] undefined (as instruction cannot be byte aligned).
- **When the processor is executing in Jazelle state:**
  - All instructions are 8 bits wide
  - Processor performs a word access to read 4 instructions at once

ARM

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>

This can only be done in ARM state.

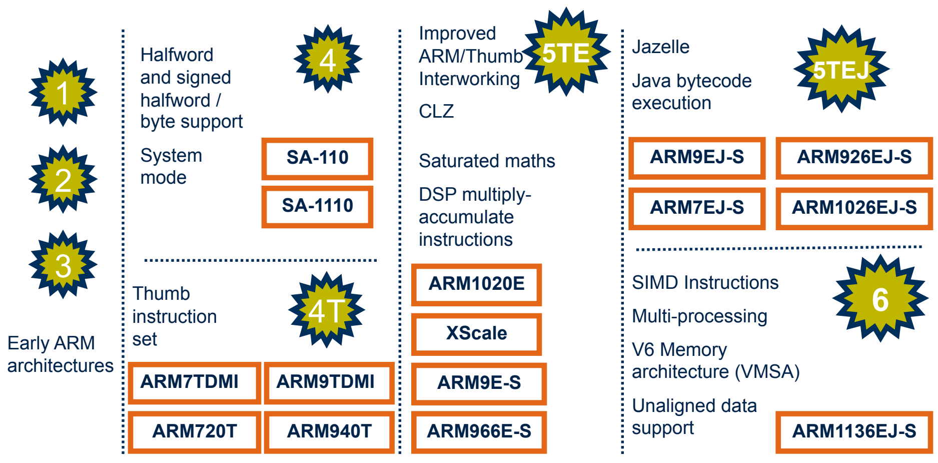
	⋮
0x1C	**FIQ**
0x18	**IRQ**
0x14	**(Reserved)**
0x10	**Data Abort**
0x0C	**Prefetch Abort**
0x08	**Software Interrupt**
0x04	**Undefined Instruction**
0x00	**Reset**

Vector Table

Vector table can be at `0xFFFF0000` on ARM720T and on ARM9/10 family devices

ARM

Development of the ARM Architecture



ARM

Agenda

ARM

Conditional Execution and Flags

- **ARM instructions can be made to execute conditionally by postfixing them with the appropriate condition code field.**
  - This improves code density *and* performance by reducing the number of forward branch instructions.

```
  CMP    r3,#0                            CMP    r3,#0
   BEQ    skip                             ADDNE  r0,r1,r2
   ADD    r0,r1,r2
skip
```

- **By default, data processing instructions do not affect the condition code flags but the flags can be optionally set by using “S”. CMP does not need “S”.**

```
loop
   …
   SUBS r1,r1,#1        decrement r1 and set flags
   BNE  loop            if Z flag clear then branch
```

Condition Codes

- **The possible condition codes are listed below:**
  - Note AL is the default and does not need to be specified

Suffix	Description	Flags tested
`EQ`	Equal	**Z=1**
`NE`	Not equal	**Z=0**
`CS/HS`	Unsigned higher or same	**C=1**
`CC/LO`	Unsigned lower	**C=0**
`MI`	Minus	**N=1**
`PL`	Positive or Zero	**N=0**
`VS`	Overflow	**V=1**
`VC`	No overflow	**V=0**
`HI`	Unsigned higher	**C=1 & Z=0**
`LS`	Unsigned lower or same	**C=0 or Z=1**
`GE`	Greater or equal	**N=V**
`LT`	Less than	**N!=V**
`GT`	Greater than	**Z=0 & N=V**
`LE`	Less than or equal	**Z=1 or N=!V**
`AL`	Always	

Examples of conditional execution

- **Use a sequence of several conditional instructions**

```
if (a==0) func(1);
    CMP       r0,#0
    MOVEQ     r0,#1
    BLEQ      func
```

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

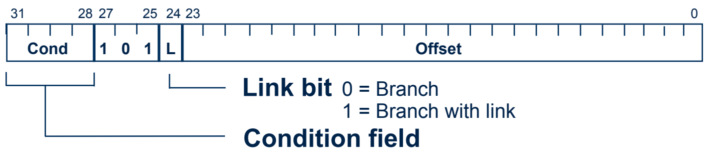
```
if (a==0) x=0;
if (a>0)  x=1;
    CMP       r0,#0
    MOVEQ     r1,#0
    MOVGT     r1,#1
```

- **Use conditional compare instructions**

```
if (a==4 || a==10) x=0;
    CMP       r0,#4
    CMPNE     r0,#10
    MOVEQ     r1,#0
```

Branch instructions

- **Branch :** `B{<cond>} label`
- **Branch with Link :** `BL{<cond>} subroutine_label`



- **The processor core shifts the offset field left by 2 positions, sign-extends it and adds it to the PC**
  - ± 32 Mbyte range
  - How to perform longer branches?

Data processing Instructions

- **Consist of :**

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

- **These instructions only work on registers, NOT memory.**
- **Syntax:**

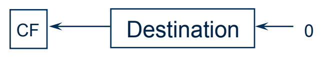
```
<Operation>{<cond>}{S} Rd, Rn, Operand2
```

  - Comparisons set flags only - they do not specify Rd
  - Data movement does not specify Rn

- **Second operand is sent to the ALU via barrel shifter.**

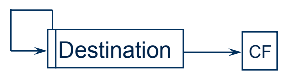
The Barrel Shifter

LSL : Logical Left Shift



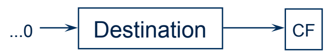
Multiplication by a power of 2

ASR: Arithmetic Right Shift



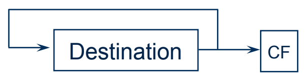
Division by a power of 2, preserving the sign bit

LSR : Logical Shift Right



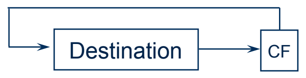
Division by a power of 2

ROR: Rotate Right



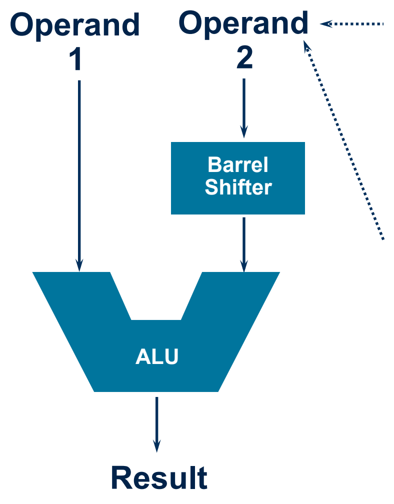
Bit rotate with wrap around from LSB to MSB

RRX: Rotate Right Extended



Single bit rotate with wrap around from CF to MSB

Using the Barrel Shifter: The Second Operand



Register, optionally with shift operation

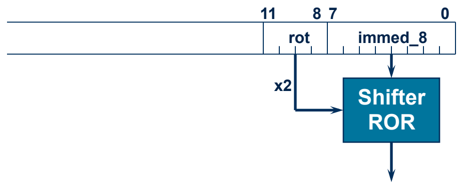
- Shift value can be either be:
  - 5 bit unsigned integer
  - Specified in bottom byte of another register.
- Used for multiplication by constant

Immediate value

- 8 bit number, with a range of 0-255.
  - Rotated right through even number of positions
- Allows increased range of 32-bit constants to be loaded directly into registers

Immediate constants (1)

- **No ARM instruction can contain a 32 bit immediate constant**
  - All ARM instructions are fixed as 32 bits long
- **The data processing instruction format has 12 bits available for operand2**

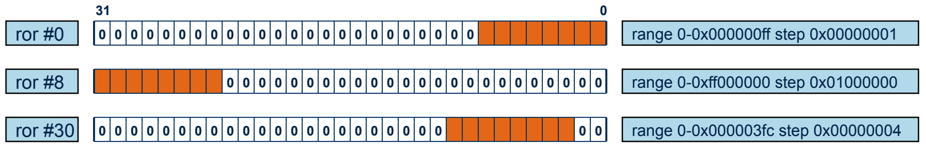


Quick Quiz:
`0xe3a004ff`
`MOV r0, #???`

- **4 bit rotate value (0-15) is multiplied by two to give range 0-30 in steps of 2**
- **Rule to remember is "8-bits shifted by an even number of bit positions".**

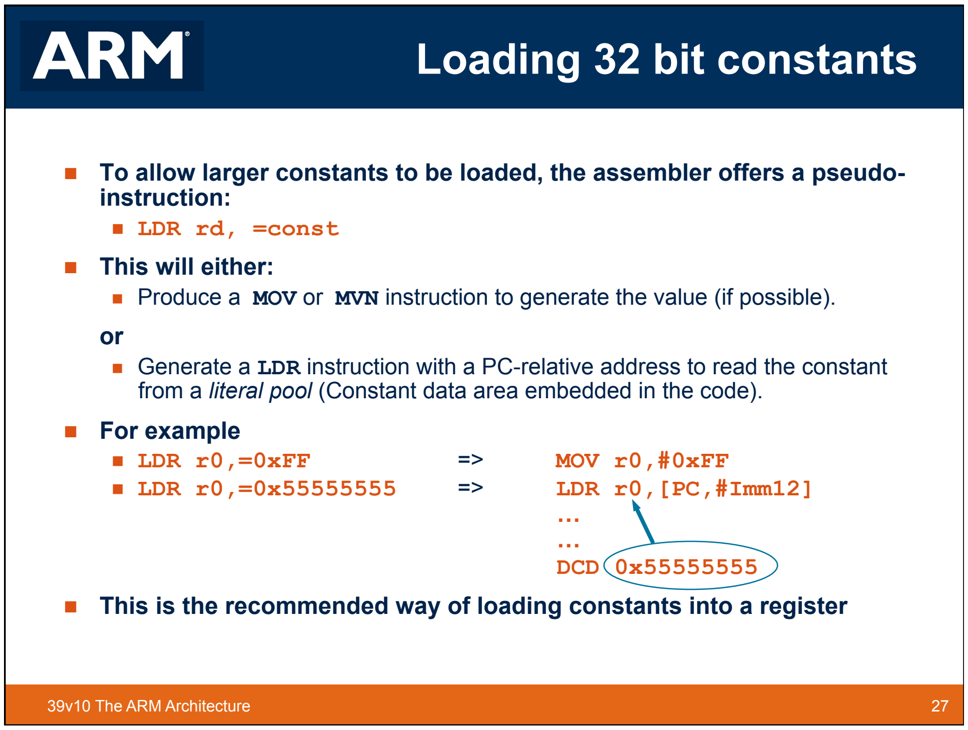
Immediate constants (2)

- **Examples:**



31
0
ror #0
range 0-0x000000ff step 0x00000001
ror #8
range 0-0xff000000 step 0x01000000
ror #30
range 0-0x000003fc step 0x00000004

- **The assembler converts immediate values to the rotate form:**
  - `MOV r0,#4096          ; uses 0x40 ror 26`
  - `ADD r1,r2,#0xFF0000 ; uses 0xFF ror 16`
- **The bitwise complements can also be formed using MVN:**
  - `MOV r0, #0xFFFFFFFF        ; assembles to MVN r0,#0`
- **Values that cannot be generated in this way will cause an error.**



Loading 32 bit constants

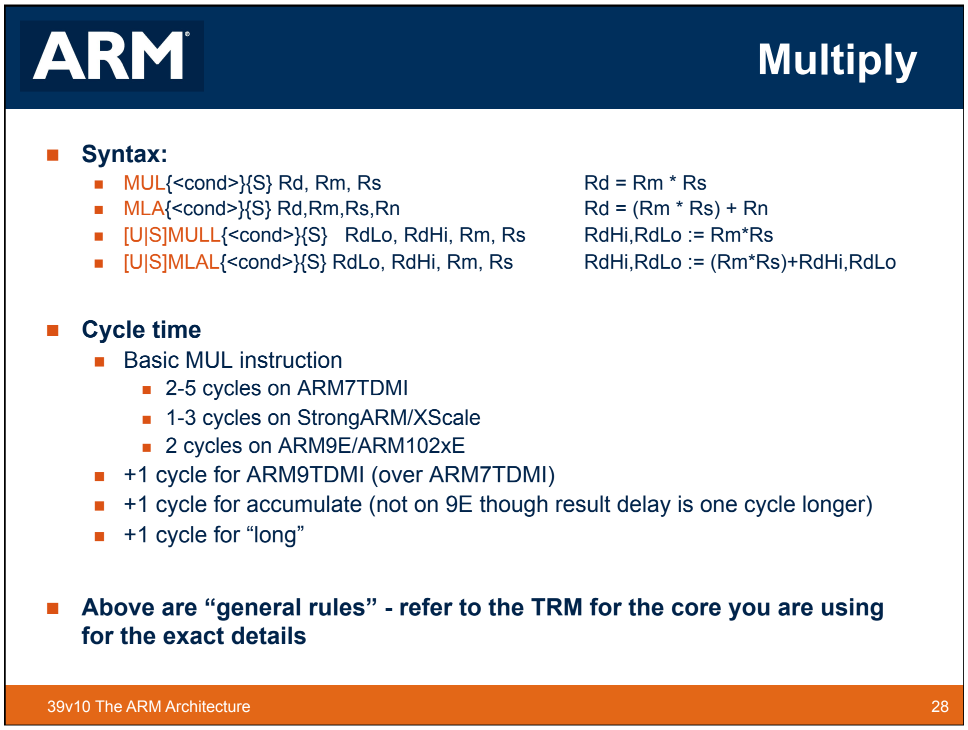
- **To allow larger constants to be loaded, the assembler offers a pseudo-instruction:**
  - `LDR rd, =const`
- **This will either:**
  - Produce a **`MOV`** or **`MVN`** instruction to generate the value (if possible).

  or
  - Generate a **`LDR`** instruction with a PC-relative address to read the constant from a *literal pool* (Constant data area embedded in the code).
- **For example**

```
LDR r0,=0xFF          =>      MOV r0,#0xFF
LDR r0,=0x55555555    =>      LDR r0,[PC,#Imm12]
                              …
                              …
                              DCD 0x55555555
```

- **This is the recommended way of loading constants into a register**

39v10 The ARM Architecture 27



Multiply

- **Syntax:**

MUL{<cond>}{S} Rd, Rm, Rs	Rd = Rm * Rs
MLA{<cond>}{S} Rd,Rm,Rs,Rn	Rd = (Rm * Rs) + Rn
[U|S]MULL{<cond>}{S} RdLo, RdHi, Rm, Rs	RdHi,RdLo := Rm*Rs
[U|S]MLAL{<cond>}{S} RdLo, RdHi, Rm, Rs	RdHi,RdLo := (Rm*Rs)+RdHi,RdLo

- **Cycle time**
  - Basic MUL instruction
    - 2-5 cycles on ARM7TDMI
    - 1-3 cycles on StrongARM/XScale
    - 2 cycles on ARM9E/ARM102xE
  - +1 cycle for ARM9TDMI (over ARM7TDMI)
  - +1 cycle for accumulate (not on 9E though result delay is one cycle longer)
  - +1 cycle for “long”
- **Above are “general rules” - refer to the TRM for the core you are using for the exact details**

39v10 The ARM Architecture 28

Single register data transfer

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

- **Memory system must support all access sizes**
- **Syntax:**
  - `LDR`{<cond>}{<size>} Rd, <address>
  - `STR`{<cond>}{<size>} Rd, <address>

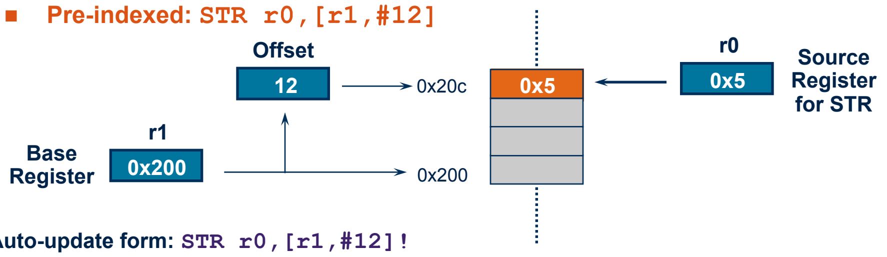
  e.g. `LDREQB`

Address accessed

- **Address accessed by LDR/STR is specified by a base register plus an offset**
- **For word and unsigned byte accesses, offset can be**
  - An unsigned 12-bit immediate value (ie 0 - 4095 bytes).
    ```
    LDR r0,[r1,#8]
    ```
  - A register, optionally shifted by an immediate value
    ```
    LDR r0,[r1,r2]
    LDR r0,[r1,r2,LSL#2]
    ```
- **This can be either added or subtracted from the base register:**
  ```
  LDR r0,[r1,#-8]
  LDR r0,[r1,-r2]
  LDR r0,[r1,-r2,LSL#2]
  ```
- **For halfword and signed halfword / byte, offset can be:**
  - An unsigned 8 bit immediate value (ie 0-255 bytes).
  - A register (unshifted).
- **Choice of *pre-indexed* or *post-indexed* addressing**

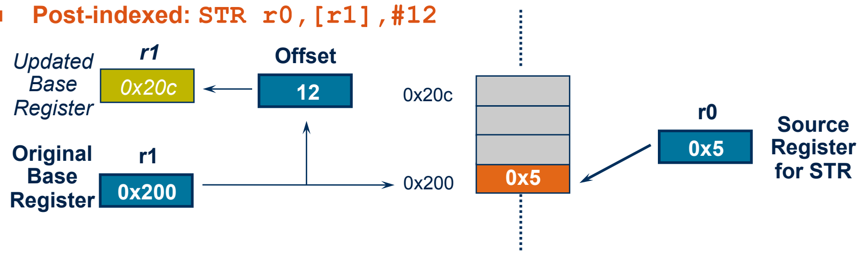
Pre or Post Indexed Addressing?

- **Pre-indexed:** `STR r0,[r1,#12]`



Auto-update form: `STR r0,[r1,#12]!`

- **Post-indexed:** `STR r0,[r1],#12`

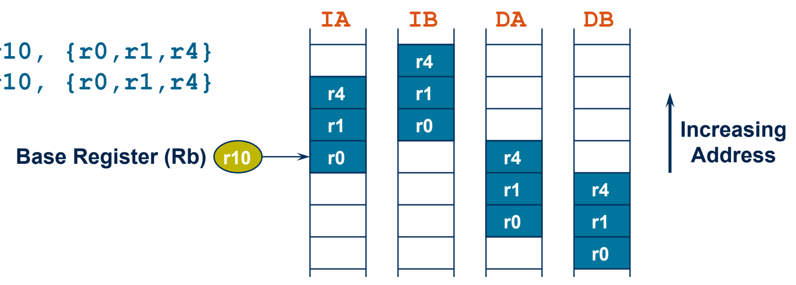


LDM / STM operation

- **Syntax:**
  `<LDM|STM>`{<cond>}<addressing_mode> Rb{!}, <register list>
- **4 addressing modes:**

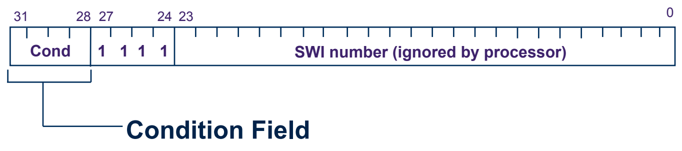
`LDMIA / STMIA`	increment after
`LDMIB / STMIB`	increment before
`LDMDA / STMDA`	decrement after
`LDMDB / STMDB`	decrement before

```
LDMxx r10, {r0,r1,r4}
STMxx r10, {r0,r1,r4}
```



ARM

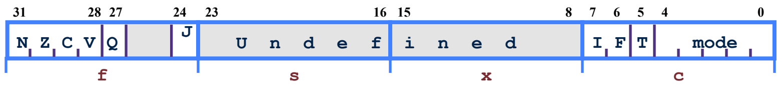
Software Interrupt (SWI)



- 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.
- Syntax:
  - `SWI{<cond>} <SWI number>`

ARM

PSR Transfer Instructions

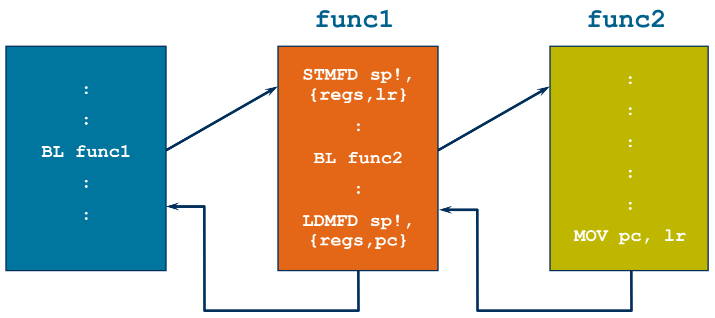


- MRS and MSR allow contents of CPSR / SPSR to be transferred to / from a general purpose register.
- Syntax:
  - `MRS{<cond>} Rd,<psr>          ; Rd = <psr>`
  - `MSR{<cond>} <psr[_fields]>,Rm ; <psr[_fields]> = Rm`

where
  - `<psr> = CPSR or SPSR`
  - `[_fields] = any combination of 'fsxc'`
- Also an immediate form
  - `MSR{<cond>} <psr_fields>,#Immediate`
- In User Mode, all bits can be read but only the condition flags (_f) can be written.

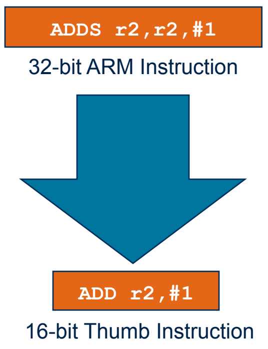
ARM Branches and Subroutines

- **B <label>**
  - PC relative. ±32 Mbyte range.
- **BL <subroutine>**
  - Stores return address in LR
  - Returning implemented by restoring the PC from LR
  - For non-leaf functions, LR will have to be stacked



Thumb

- **Thumb is a 16-bit instruction set**
  - Optimised for code density from C code (~65% of ARM code size)
  - Improved performance from narrow memory
  - Subset of the functionality of the ARM instruction set
- **Core has additional execution state - Thumb**
  - Switch between ARM and Thumb using **BX** instruction

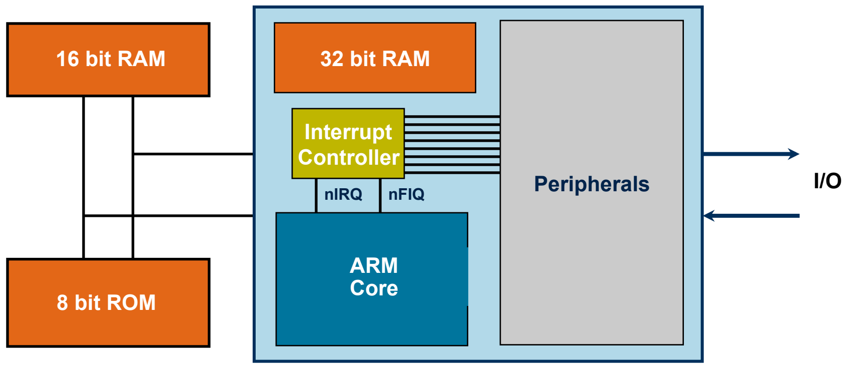


For most instructions generated by compiler:

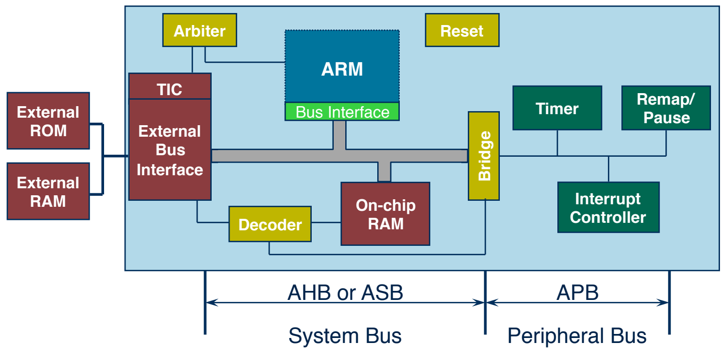
- Conditional execution is not used
- Source and destination registers identical
- Only Low registers used
- Constants are of limited size
- Inline barrel shifter not used

Agenda

Example ARM-based System

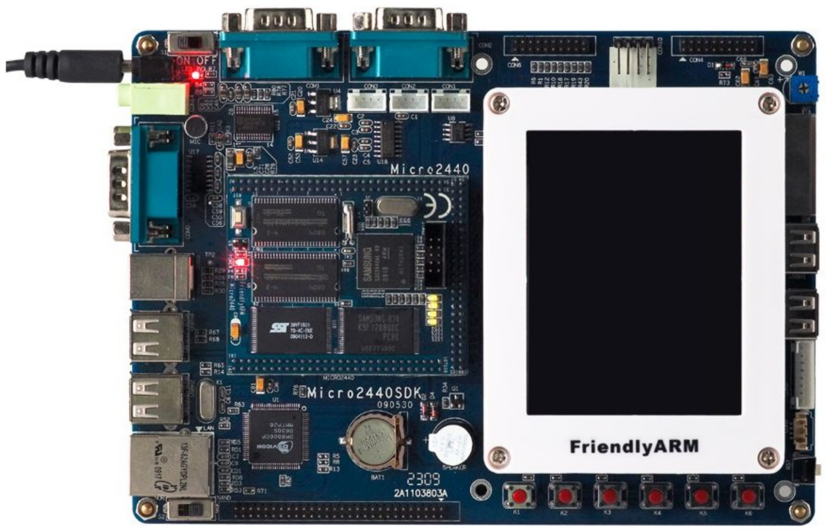


AMBA

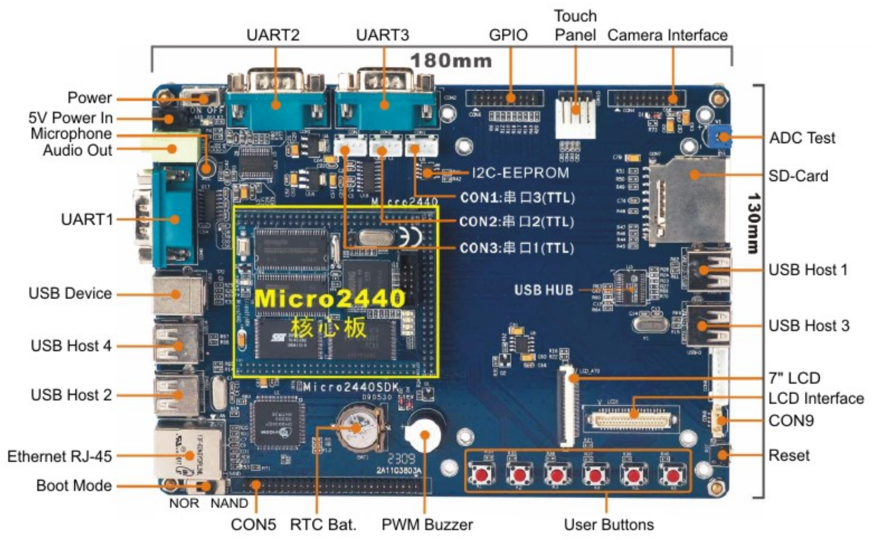


- **AMBA**
  - Advanced Microcontroller Bus Architecture
- **ADK**
  - Complete AMBA Design Kit
- **ACT**
  - AMBA Compliance Testbench
- **PrimeCell**
  - ARM's AMBA compliant peripherals

FriendlyArm micro2440-35



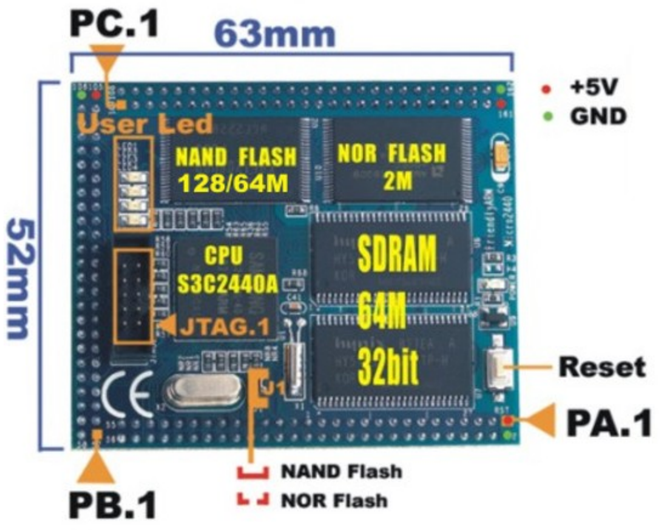
FriendlyArm micro2440



Specification: SDK-Board

- **Dimension: 180 x 130 mm**
- **EEPROM: 1024 Byte (I2C)**
- **Ext. Memory: SD-Card socket**
- **Serial Ports: 3x DB9 connector (RS232)**
- **USB: 4x USB-A Host, 1x USB-B Device**
- **Audio Output: 3.5 mm stereo jack**
- **Audio Input: 3.5mm jack (mono) + Condenser microphone**
- **Ethernet: RJ-45 10/100M (DM9000)**
- **RTC: Real Time Clock with battery**
- **Beeper: PWM buzzer**
- **Camera: 20 pin Camera interface (2.0 mm)**
- **LCD: 41 pin connector for FriendlyARM Displays (3.5" and 7") and VGA Board**
- **Touch Panel: 4 pin**
- **User Inputs: 6x push buttons and 1x A/D pot**
- **Expansion headers (2.0 mm)**
- **Power: 5V connector, power switch and LED**
- **Power Supply: regulated 5V**

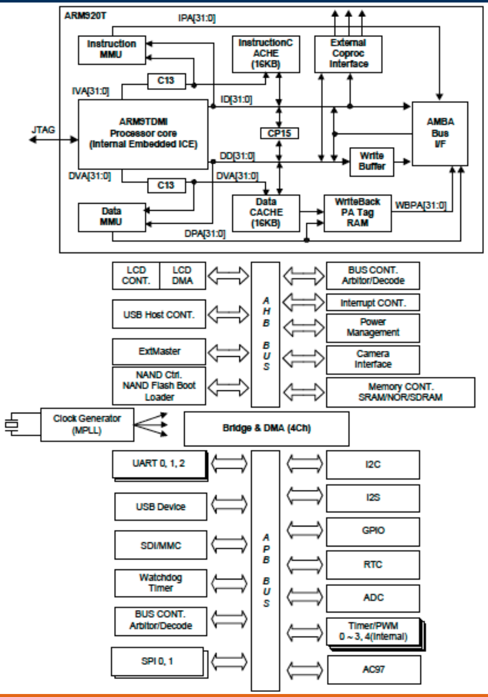
Stamp Module



Specification: Stamp Module

- **Dimension: 63 x 52 mm**
- **CPU: 400 MHz Samsung S3C2440A ARM920T (max freq. 533 MHz)**
- **RAM: 64 MB SDRAM, 32 bit Bus**
- **Flash: 64 MB / 128 MB / 256 MB / 1GB NAND Flash and 2 MB NOR Flash with BIOS**
- **LCD Interface**
  - STN Displays:
    - Monochrome, 4 gray levels, 16 gray levels, 256 colors, 4096 colors
    - Max: 1024x768
  - TFT Displays:
    - Monochrome, 4 gray levels, 16 gray levels, 256 colors, 64k colors, true color
    - Max: 1024x768
- **Touch Panel: 4 wire resistive**
- **User Outputs: 4x LEDs**
- **Expansion headers (2.0 mm)**
- **Debug: 10 pin JTAG (2.0 mm)**
- **OS Support**
  - Windows CE 5 and 6
  - Linux 2.6
  - Android

- FriendlyArm Processor
- Samsung S3C2440A
  ARM920T



ARM920T
IPA[31:0]
Instruction MMU
InstructionCACHE (16KB)
External Coproc Interface
C13
IVA[31:0]
ID[31:0]
ARM9TDMI Processor core (Internal Embedded ICE)
JTAG
CP15
AMBA Bus I/F
DD[31:0]
Write Buffer
DVA[31:0]
C13
DVA[31:0]
Data MMU
Data CACHE (16KB)
WriteBack PA Tag RAM
WBPA[31:0]
DPA[31:0]
LCD CONT.
LCD DMA
USB Host CONT.
ExtMaster
NAND Ctrl. NAND Flash Boot Loader
AHB BUS
BUS CONT. Arbitor/Decode
Interrupt CONT.
Power Management
Camera Interface
Memory CONT. SRAM/NOR/SDRAM
Clock Generator (MPLL)
Bridge & DMA (4Ch)
UART 0, 1, 2
USB Device
SDI/MMC
Watchdog Timer
BUS CONT. Arbitor/Decode
SPI 0, 1
APB BUS
I2C
I2S
GPIO
RTC
ADC
Timer/PWM 0 ~ 3, 4(Internal)
AC97