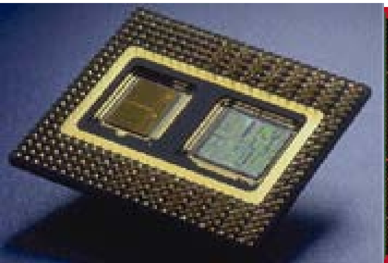
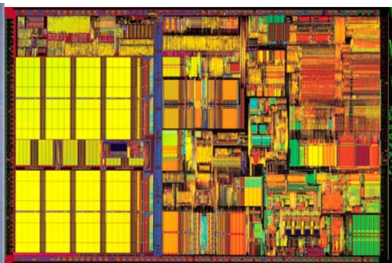
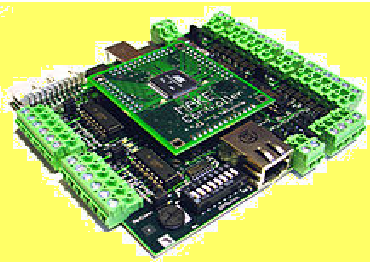
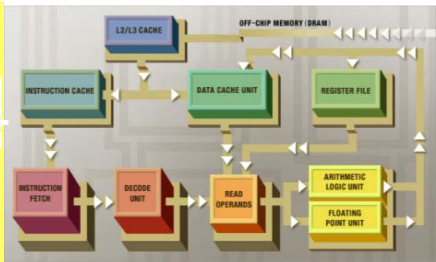
Lecture 15
ARM Processor – A RISC Architecture

Bryan Ackland
Department of Electrical and Computer Engineering
Stevens Institute of Technology
Hoboken, NJ 07030

Adapted from HCS12/9S12 An Introduction to Software and Hardware Interfacing Han-Way Huang, 2010

What Makes a Good Instruction Set ?

- Supply functions that are useful to programmer
  - taking into account frequency of use
- Efficient implementation in terms of hardware
  - logic, registers and memory
- Backward compatibility (think about x86)
- Good compiler target
  - high level languages provide data and process abstraction and support structured programming which improves reliability and verifiability of software and shortens development time
  - compiler bridges semantic gap between high-level language and machine instructions
  - want architecture for which compiled code rivals efficiency & performance of assembly code
- High performance
  - how much work can processor do in given period of time

Instruction Set Complexity

- Prior to 1980, computer architects used increasing power of VLSI (integrated circuits) to provide instructions of increasing complexity
  - each instruction performing a complex sequence of operations over many clock cycles
  - processors were often marketed in terms of how much could be accomplished in single instruction and how many addressing modes
  - CPU was itself a micro-coded engine in which each machine instruction was implemented as sequence of microcode instructions stored in high speed microcode ROM
  - some architectures even allowed programmers to extend instruction set to do application specific operations by writing their own microcode.
  - difficult to target the most complex instructions from compiler (e.g. VAX has polynomial evaluation and queue insertion instructions)

RISC Architectures

- How can we improve microprocessor performance?

1. Use a large number of complicated and powerful instructions to do more work with each instruction
   - historical approach

2. Use small, highly optimized instructions to do less work per instruction but execute them much faster
   - championed by Berkeley RISC project (Patterson & Sequin) 1980
   - **R**educed **I**nstruction **S**et **C**omputer
   - doesn't mean reduced # of instructions
   - means reduced complexity of instructions

- Alternative (historical) approach became known as CISC
  - **C**omplex **I**nstruction **S**et **C**omputer

Evolution of Microprocessor Architecture

- Since 1980, computer architects used increasing power of VLSI (integrated circuits) to add architectural features (originally developed for use on large mainframes) to microprocessors
  - **Pipelining:** execute instruction in stages (e.g. fetch, decode, execute, store). Start next instruction once current instruction has completed first stage. Allows for faster clock and overlapped execution
  - **Cache Memory:** a small fast memory located close to CPU that holds most recently accessed code or data
  - **Super-scalar execution:** execute multiple instructions in parallel by dispatching data to multiple functional units (ALU, multiplier etc.)
  - **Pre-fetch and Branch prediction:** guess whether a branch will be taken and pre-fetch instructions based on that guess
- Each of these is either easier to implement or provides greater performance impact in RISC architecture

CISC vs. RISC

CISC processor	RISC Processor
Variable length instructions with many formats	Fixed instruction size with uniform instruction format
Memory locations can be used as arithmetic operands. Rich set of addressing modes	Load/store architecture where arithmetic instructions operate only on registers. Simple addressing modes
Small register bank with most registers having specific purpose	Large general purpose register bank
Instruction decoded using microcode sequences in ROM	Hard-wired instruction decode logic
Complex data types supported in hardware (strings, complex numbers)	Few data types supported in hardware
Many clock cycles per instruction	Single-cycle execution
Little overlap between instructions	Pipelined execution

So who won?

- Highly successful architectures of both types:

RISC	CISC
SPARC (SUN)	x86/Pentium (Intel)
PowerPC (Motorola, IBM)	MC68000 (Motorola / Freescale)
ARM (by license)	HCS12 (Motorola / Freescale)
MIPS (by license)	PDP/VAX (DEC)

- Once an instruction set architecture has been defined and released as a product, backward compatibility limits scope of changes to architecture
- Over the years, the line between RISC and CISC has blurred with each moving to “middle ground” to improve performance.
  - RISC chips have leveraged improvements in VLSI to develop more complex instruction sets that still run at very high speed
  - CISC chips have leveraged improvements in VLSI to incorporate parallelism (pipelining, super-scalar, multicore) into their architectures

ARM Processor

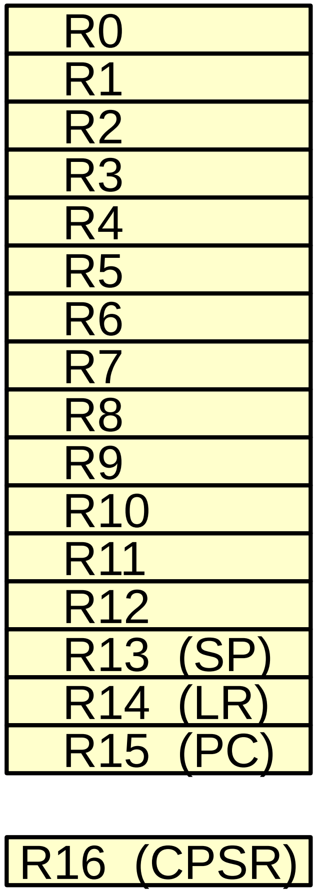
- ARM is short for **A**dvanced **R**ISC **M**achines
- Founded in 1990 by Acorn (U.K.), Apple & VLSI Technology
  - goal was to develop high performance low power microprocessor for embedded applications
- ARM does not make microprocessors
  - Intellectual Property (IP) supplier
  - microprocessor cores, standard cells, graphics & multimedia engines
- Industry's leading supplier of 16/32 bit embedded RISC processors
  - over 90% of embedded 32-bit processors
  - over 20 billion ARM cores shipped in products (smart phones, PDA's, digital cameras etc.)
  - family of processors ARM6, ARM7, ARM9, ARM10, ARM11

ARM Architecture

- 32-bit RISC processor core
- 32-bit address and data busses
- Fixed length 32-bit instruction
- 3-stage pipeline (ARM7) and support for cache
- 8-bit and 32-bit data types
  - data operations (arithmetic) are all 32-bit
  - supports 8-bit and 32-bit data transfer
- Load/store architecture
  - does not support data operations directly on memory locations
  - data operands must first be loaded into registers and then stored back into memory to save the results
- Every instruction can be conditionally executed
- Three operand data operations with optional multi-bit shift
- Most instructions executed in single cycle

ARM Register Set

- Total of 37 32-bit registers
- 17 visible at any one time
  - depends on operating mode
  - normal code runs in user mode
  - other modes include interrupt mode and supervisor mode (for operating system calls)
  - other modes have their own registers to minimize data save instructions
- R0-R12 are general purpose registers
- R13 is used as stack pointer (SP)
- R14 is subroutine link register
  - holds return address
- R15 is program counter
- R16 is current program status register
  - holds condition code bits N, Z, C and V



ARM Instruction Set

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Cond				0	0	I	Opcode				S	Rn				Rd				Operand2												***Data processing***
Cond				0	0	0	0	0	0	A	S	Rd				Rn				Rs				1	0	0	1	Rm				***Multiply***
Cond				0	0	0	0	1	U	A	S	RdHi				RdLo				Rs				1	0	0	1	Rm				***Long Multiply***
Cond				0	1	I	P	U	B	W	L	Rn				Rd				Offset												***Load/Store***
Cond				1	0	0	P	U	S	W	L	Rn				Register List																***Ld/St Multiple***
Cond				1	0	1	L	Offset																								***Branch***

- Uniform instruction coding
  - opcode always in same bit position
  - many fields have same meaning across different instruction types
- Allows faster instruction decoding

Conditional Execution

- Most instruction sets only allow branches to be executed conditionally.
- Many branches skip over one or two instructions
- In ARM, all instructions are conditional
- This removes the need for many branches, which stall the pipeline (3 cycles to refill).
- Allows very dense in-line code, without branches.

HCS12

```
        …
        bne     skip
        inc     total
skip:   clra
        …
```

ARM

```
addeq   r3, r3, #1
sub     r0, r0, r0
…
```

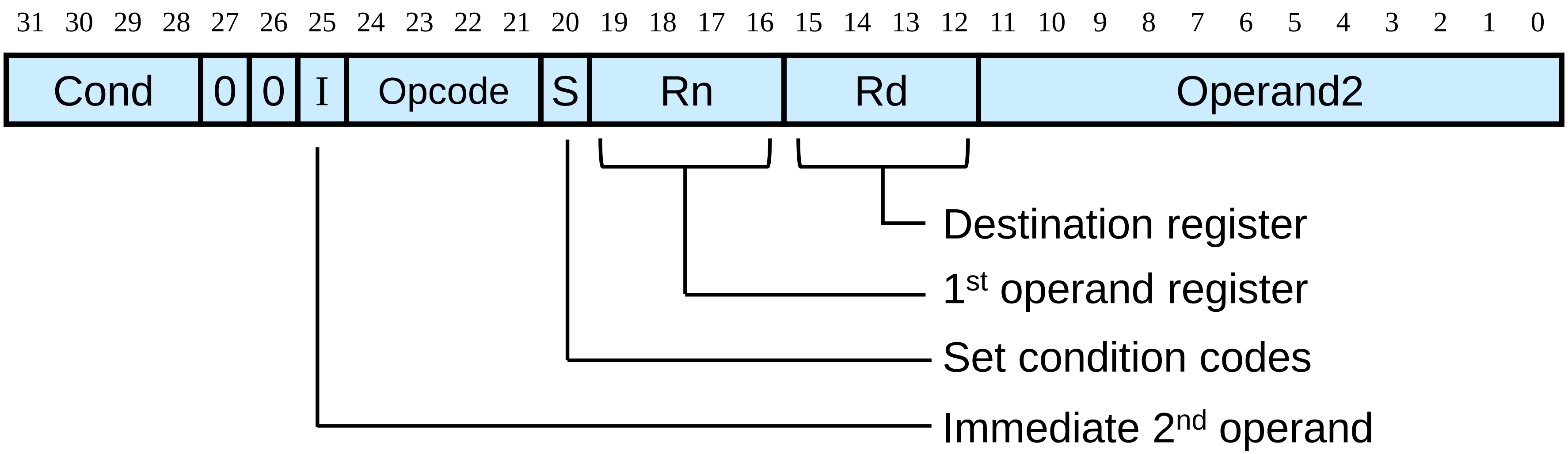
Conditional Codes

- 14 available conditions
  – Normal (unconditional) instructions use code AL

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 and Z set	unsigned lower or same
1010	GE	N equals V	greater or equal
1011	LT	N not equal V	less than
1100	GT	Z clear and N equals V	greater than
1101	LE	Z set or (N not equal V)	less than or equal
1110	AL	---	**always**

Data Processing Instructions

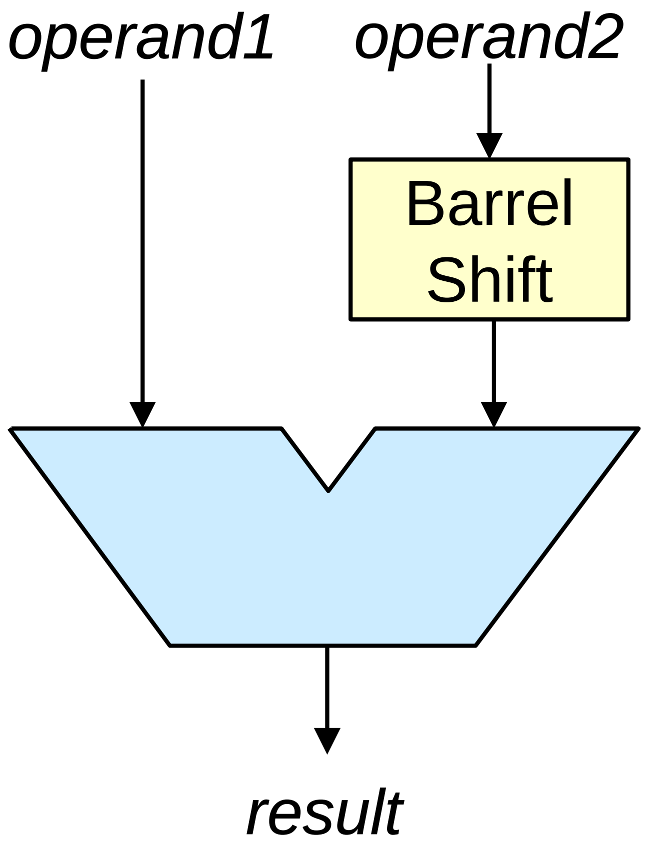
- ARM data processing instructions specify up to 3 registers
  - Destination (result) register plus two operand registers
  - no memory locations – only registers
  - Immediate bit and update condition code bit



- Condition codes are only set if S bit is ‘1’
- Operand2 contains either:
  - register address (if I = ‘0’) OR
  - immediate value (if I = ‘1’)
  - together with a shift specification

Barrel Shifter

- ALU includes a barrel shifter than can shift operand 2 up to 32 bits to the left or right



- Operand 2 can either be:

- **Register** shifted by
  - immediate constant OR
  - value in another register
  - logical / arithmetic / rotate
  - left/right

- 8-bit **immediate value**
  - rotated right through an even number of positions (2-32)

Data Processing Opcodes

Opcode	Mnemonic	Flags
ARITH	ADD	operand1 + operand2
	ADC	operand1 + operand2 + carry
	SUB	operand1 – operand2
	SBC	operand1 – operand2 – carry + 1
	RSB	operand2 – operand1
	RSC	operand2 – operand1 – carry + 1
LOGIC	AND	operand1 AND operand2
	EOR	operand1 EXOR operand2
	ORR	operand1 OR operand2
	BIC	operand1 AND NOT operand2
TEST	CMP	same as SUB but result not written
	CMN	same as ADD but result not written
	TST	same as AND but result not written
	TEQ	same as EOR but result not written
MOVE	MOV	operand2 (operand1 is ignored)
	MVN	NOT operand2 (operand1 is ignored)

Data Processing Examples

- ADD r0, r1, r2 ; r0 = r1 + r2
- SUBGT r3, r3, #1 ; r3 = r3 – 1 if GT true
- RSBLES r4, r4, #5 ; r4 = 5 – r4 if LE & set CC's
- TSTEQ r2, #6 ; if Z=0, form (r2 AND #6) & set CC's
- AND r0, r1, r2 ; r0 = r1 AND r2
- BICHI r2, r3, #7 ; if HI, r2 = r3 with 3 LSBits set to 0
- MVNEQ r1, #0 ; if Z=1, set r1 = -1
- ADD r1, r0, r0, LSL #2 ; r1 = r0 + (r0*4)
- MOV r3, #0x40, ROR #26 ; set r3 = 4096

Multiply Instruction

- ARM does signed/unsigned 32 x 32 multiply
  - produces signed/unsigned least significant 32-bit result

31 30 29 28	27	26	25	24	23	22	21	20	19 18 17 16	15 14 13 12	11 10 9 8	7	6	5	4	3 2 1 0
Cond	0	0	0	0	0	0	A	S	Rd	Rn	Rs	1	0	0	1	Rm

- MUL{<cond>}{S} Rd, Rm, Rs ; Rd = Rm * Rs

- If A bit is set, we get signed/unsigned multiply accumulate:

- MULA{<cond>}{S} Rd, Rn, Rm, Rs ; Rd = (Rm * Rs) + Rn

- Multiply does not normally complete in one cycle
  - cycle count depends on implementation
  - early termination if only ‘0’s left in multiplier

Long Multiply Instruction

- Produces signed and unsigned 64-bit result

31–28	27	26	25	24	23	22	21	20	19–16	15–12	11–8	7	6	5	4	3–0
Cond	0	0	0	0	1	U	A	S	RdHi	RdLo	Rs	1	0	0	1	Rm

- Multiply long:

  RdHi:RdLo = Rm * Rs

- Multiply accumulate long:

  RdHi:RdLo = Rm * Rs + RdHi:RdLo

- Available in signed and unsigned versions:

  UMULL{<cond>}{S} RdLo,RdHi,Rm,Rs

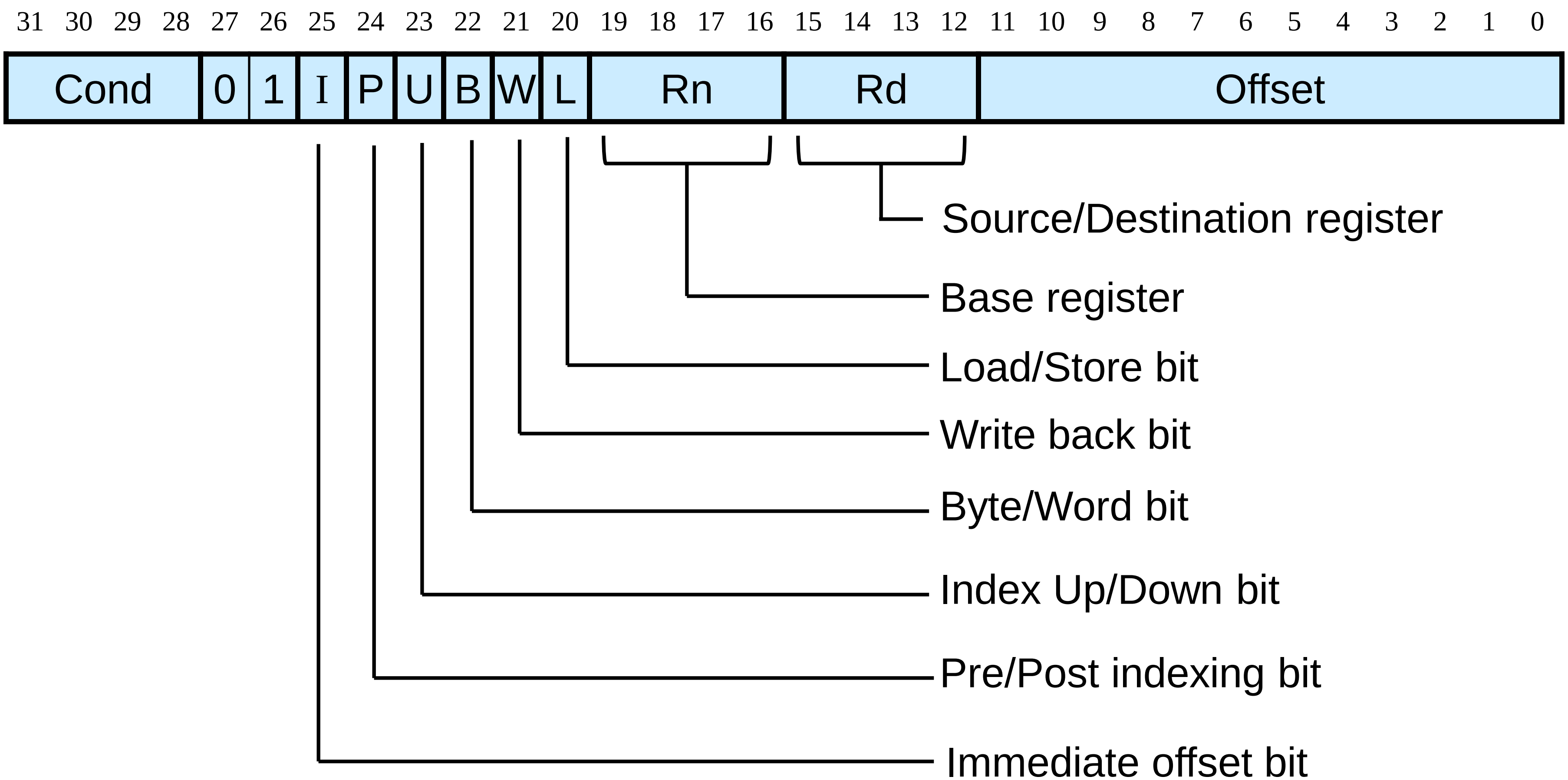
  UMLAL{<cond>}{S} RdLo,RdHi,Rm,Rs

  SMULL{<cond>}{S} RdLo,RdHi,Rm,Rs

  SMLAL{<cond>}{S} RdLo,RdHi,Rm,Rs

Load/Store Instructions

- These simply move data between registers and memory



- All load/stores can be conditionally executed

Load/Store Format

- LDR load register with word from memory
- LDRB load register with byte from memory
- STR store register to word in memory
- STRB store register to byte in memory

<LDR|STR>{<cond>}{<size>} Rd, <address>

- Memory address is formed using variety of addressing modes
- All address modes are indirect via register
  - no extended (direct addressing mode) since cannot fit 32-bit address into instruction
  - no immediate addressing mode (constants must be loaded into memory within offset distance of PC)

Memory Addressing Modes

- Register indirect addressing

```
LDR   r0, [r1]              ; load r0 with contents of memory
                            ; pointed to by r1
```

- Base plus immediate index addressing

```
LDR   r0, [r1, #2]          ; load r0 with contents of memory
                            ; located at address [r1]+2
```

- Base plus register index addressing

```
STR   r0, [r1, r2, LSL #2]  ; store r0 to memory location
                            ; whose address is [r1] + ([r2]<<2)
```

- Auto increment pre-index addressing

```
LDR   r0, [r1, #4]!         ; load r0 with contents of memory
                            ; located at address [r1]+4 and update
                            ; r1 to new address
```

- Auto increment post-index addressing

```
LDR   r0, [r1], #4          ; load r0 with contents of memory
                            ; located at address [r1] and then
                            ; increment r1 by 4
```

Branch Instructions

- Conditional execution is good for replacing branches around small number of instructions
  - not efficient for branches involving large numbers of instructions
  - need to conditionally execute all instructions related to both branch outcomes
- ARM provides Branch (B) and Branch with Link (BL)

31–28	27	26	25	24	23–0
Cond	1	0	1	L	Offset

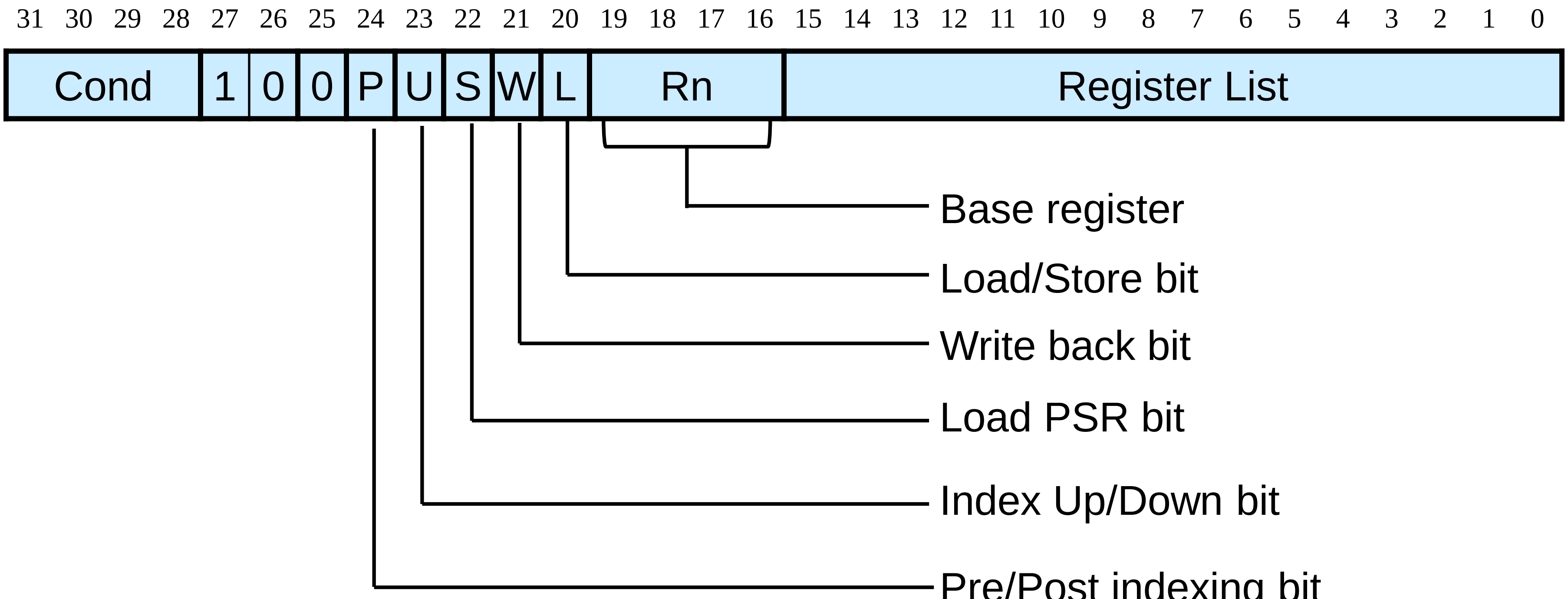
- Offset provides 24-bit signed word offset relative to PC
  - do not need byte offset since instruction are all 32-bit word aligned
- Provides branch range of $\pm 32\ Mbytes$
- Conditional branch just uses regular condition field
- Use labels, assembler calculates offset

Subroutine Call

- Branch with Link provides subroutine linkage
  - BL{cond} sub_label
- [PC] is stored in link register R14
- Return simply restores PC from link register
  - mov PC, R14
- For nested subroutine calls, programmer must save return address by moving from LR to stack
- Large number of registers could make saving and restoring registers very slow
- ARM provides Load/Store Multiple instruction (LDM/STM)

Block Data Transfer (LDM/STM)

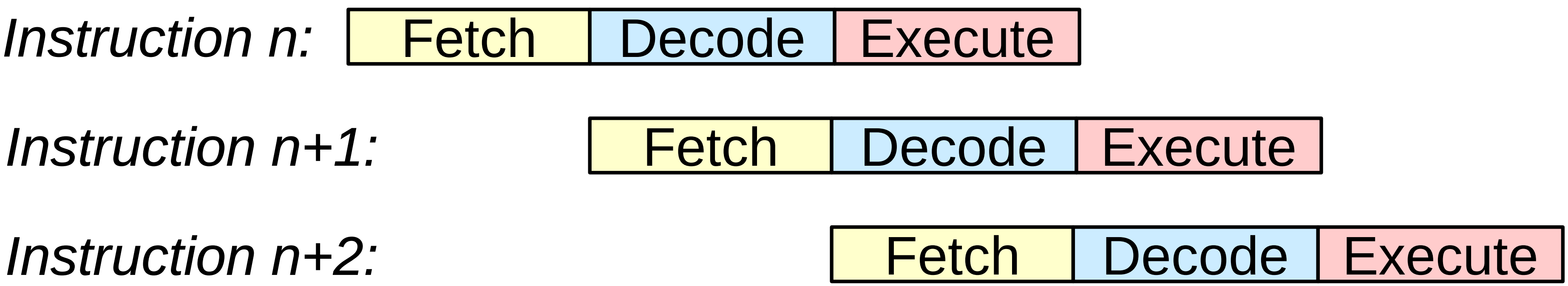
- LDM, STM: load and store any subset of registers



  - loaded from and stored into contiguous block of memory relative to a base register
  - by using SP as base register and auto-indexed addressing, we can push to or pop from stack any subset of registers

ARM Pipeline

- ARM uses a 3-stage pipeline to speed instruction execution
  - Fetch: get next instruction from memory
  - Decode: Determine operand registers and ALU operation
  - Execute: Read registers, perform ALU operation and store registers
- Allows several instructions to be executing simultaneously



- Needs “bypass” paths in CPU to avoid reading new value from a register before it has been written

ARM Processors in Embedded Systems

- As stand-alone microcontrollers
  - STMicro, Atmel, Samsung, Freescale etc.
- Embedded in Applications Specific Standard Product (ASSP)
  - Atmel: Bluetooth controller
  - Conexant: Cable modem
  - LSI Logic: Ethernet switch
  - Philips: GSM processor
  - Qualcomm: CDMA baseband
  - Samsung: Ink-jet printer
- Embedded in FPGA
  - Altera and Xilinx
  - Provide mix of software and programmable hardware
  - Altera Cyclone FPGA’s can include 800MHz dual-core ARM9 processor with 32KB instruction & data caches