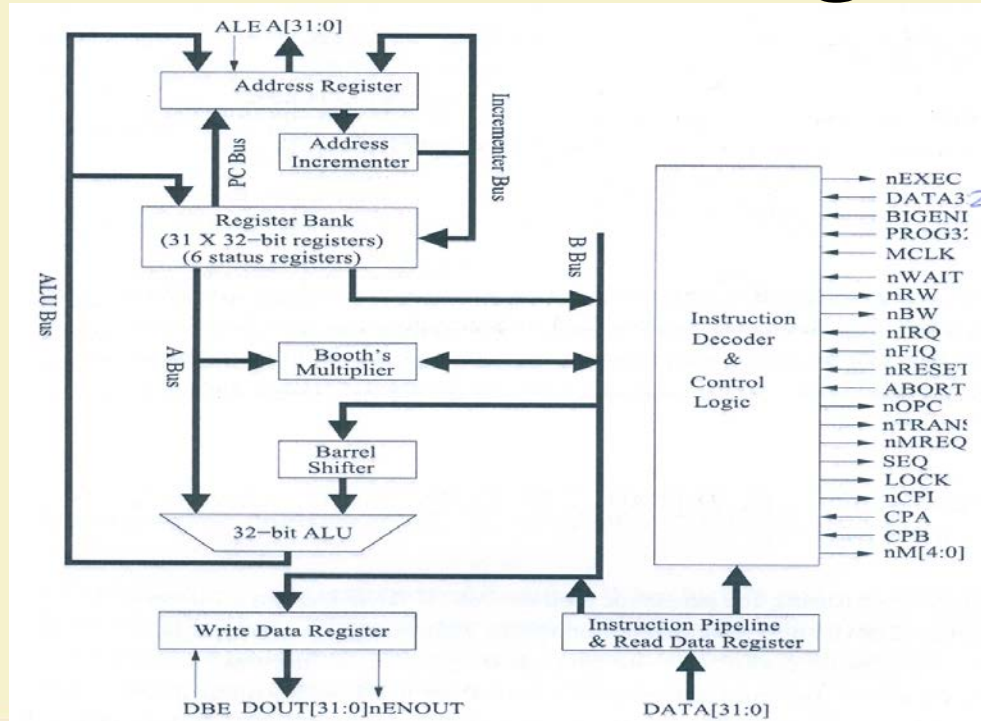


ARM7 block diagram



ARM Pipeline

- **A very important feature of ARM processors. It has different versions**
- **3-stage pipeline – ARM7TDMI and earlier**
- **5-stage pipeline – ARMS, ARM9TDMI**
- **6-stage pipeline – ARM10TDMI**
- **8-stage pipeline – ARM11**

3-stage Pipeline

- Classical *fetch-decode-execute* pipeline
- First stage reads an instruction from memory and increments the value in the instruction address register
- Next stage decodes instruction and prepares control signals to execute it
- Third stage does the actual work: reading operands from register file, performing ALU operations, writes back the modified register values

5-stage Pipeline

- In 3-stage pipeline, pipeline stall caused by every data transfer instruction – the next instruction cannot be fetched while memory is being read/written
- Instruction and data memory separated
- Register read step moved to decode stage
- Execute stage split into three – performing arithmetic computations, memory access, write result back to register file
- Balances pipeline, reducing CPI (average number of clocks per instruction)
- However, need to forward data between pipeline stages to resolve data dependencies between the stages without stalling the pipeline

6-stage Pipeline

- In ARM10 core, instruction decode is split into two pipeline stages – *decode*, *register*
- Decode stage performs decode operation
- Register stage reads the register to be used
- A separate adder introduced in execution unit to take care of multiply-accumulate instructions
- Both instruction and data buses are 64-bit

8-stage Pipeline

- **Two new features introduced in ARM11 core**
- **Shift operation has been separated into a separate pipeline stage**
- **Both instruction and data accesses are distributed across two pipeline stages**
- **Execution unit is split into three different pipelines that can concurrently operate and commit instructions out-of-order also**

Instruction Set Architecture

- **Typical RISC architecture with several enhancements to improve performance further**
- **The RISC features are as follows**
 - **Large uniform register file with 16 general purpose registers**
 - **Load/store architecture. The instructions that process data operate only on registers and are separate from instructions that access memory**
 - **Simple addressing modes**
 - **Uniform and fixed-length instruction fields. All ARM instructions are 32-bit long and most of them have a regular three operand encoding**

Improved Features

- Each instruction controls the ALU and shifter, making the instructions more powerful
- Auto-increment and auto-decrement addressing modes supported
- Multiple load/store instructions that allow to load/store upto 16 registers at once
- Conditional execution of instructions introduced. Instruction opcode is preceded by a 4-bit condition code. For the instruction to execute, the condition must be met. Eliminates small branches and thus pipeline stalls
- Arithmetic operations may or may not affect the status bits

Registers

- 16 general purpose registers R0-R15 in user mode
- R15 is the program counter, but can also be manipulated as a general purpose register
- R13 is conventionally used as the stack pointer. ARM instruction set does not have PUSH/POP instructions
- R14 is called the *link register*. When a procedure call is made, the return address is automatically placed into this register (unlike in stack). A return from the procedure can be implemented by copying R14 to R15
- *Current Program Status Register (CPSR)* contains four 1-bit condition flags – *negative, zero, carry, overflow*
- *Saved Program Status Register (SPSR)* stores a copy of CPSR in some modes of operation

Modes of Operation

- **ARM processor operates in one of the six operating modes**
 - **User mode**
 - used to run application code
 - CPSR cannot be written
 - mode can only be changed via exception generation
 - **Fast interrupt processing mode (FIQ)**
 - Supports high speed interrupt handling
 - Generally used for a single critical interrupt source
 - **Normal interrupt processing mode (IRQ)**
 - supports all other interrupt sources in a system

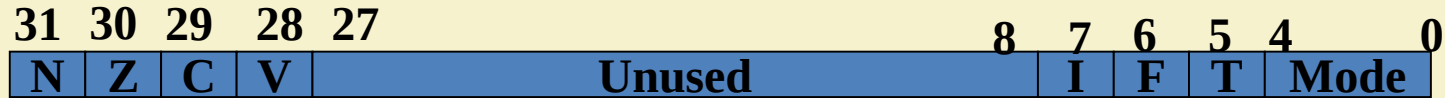
Modes of Operation (contd.)

- **Supervisor mode (SVC)**
 - entered when the processor encounters a software interrupt instruction
 - used for OS services
 - on reset, ARM enters into this mode
- **Undefined instruction mode (UNDEF)**
 - fetched instruction is not an ARM instruction or a coprocessor instruction
- **Abort mode**
 - entered in response to memory fault

ARM Registers in Different Modes

System & User	FIQ	Supervisor	Abort	IRQ	Undefined
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7_fiq	R7	R7	R7	R7
R8	R8_fiq	R8	R8	R8	R8
R9	R9_fiq	R9	R9	R9	R9
R10	R10_fiq	R10	R10	R10	R10
R11	R11_fiq	R11	R11	R11	R11
R12	R12_fiq	R12	R12	R12	R12
R13	R13_fiq	R13_svc	R13_abt	R13_irq	R13_und
R14	R14_fiq	R14_svc	R14_abt	R14_irq	R14_und
R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und

CPSR Register



N: negative

Z: Zero

C: Carry

V: Overflow

I: IRQ mode

F: FRQ mode

T: THUMB instruction set

Mode: 6 operating modes



Data Types

- **Six different data types**
 - 8-bit signed and unsigned
 - 16-bit signed and unsigned
 - 32-bit signed and unsigned
- **Supports both little-endian and big-endian format**
- **Most implementations support only little-endian**

Instruction Sets

- **Two different instruction sets**
 - **ARM : Standard 32-bit instruction set**
 - Data processing
 - Data transfer
 - Block transfer
 - Branching
 - Multiply
 - Conditional
 - Software interrupts
 - **THUMB : 16-bit compressed form**
 - Code density better than most CISC
 - Dynamic decompression in pipeline

Data Processing Instructions

- Supports range of arithmetic operations – addition, subtraction, multiplication
- Bit-wise logical operations
- All operations take 2, 32-bit operands and return a 32-bit value
- First operand and result must be register
- Second operand can be a register or an immediate value
- If second operand is a register, it can be shifted or rotated before sending to the ALU

Immediate Second Operand

- Immediate operand must be 32-bit value
- All 32-bit constants cannot be specified
- All binary ones must fall within a group of 8 adjacent bit positions on a 2-bit boundary
- More formally, a valid immediate operand n must satisfy,

$$n = i \text{ ROR } (2 * r)$$

where i is a number between 0 and 255, r is between 0 and 15

- Example: 255 ($i = 255, r = 0$), 256 ($i = 1, r = 12$)

Data Processing Instructions (contd.)

- Modification of condition flags by arithmetic instructions is optional
- Flags need not be checked right after the instruction that sets them
- Examples:
 - `ADD R1, R2, R3;` $R1 = R2 + R3$
 - `ADD R1, R2, R3, LSL #2;` $R1 = R2 + R3 * 4$
 - `ADDS R1, R2, R3, LSL #2;` $R1 = R2 + R3 * 4$ and set condition code flags

Single Register Data Transfer Instructions

- Can be used to transfer 1, 2, or 4-bytes of data between a register and a memory location
- Base plus offset mode can be used
- Both pre-indexed and post-indexed modes are available
- Offset can either be a 12-bit unsigned immediate value or a register optionally shifted by an immediate value
- Offset may be added or subtracted from the base register

Examples

LDR R0, [R8]; R0 = Memory[R8]

LDR R0, [R1, -R2]; R0 = Memory[R1 – R2]

LDR R0, [R1, #4]; R0 = Memory[R1 + 4]

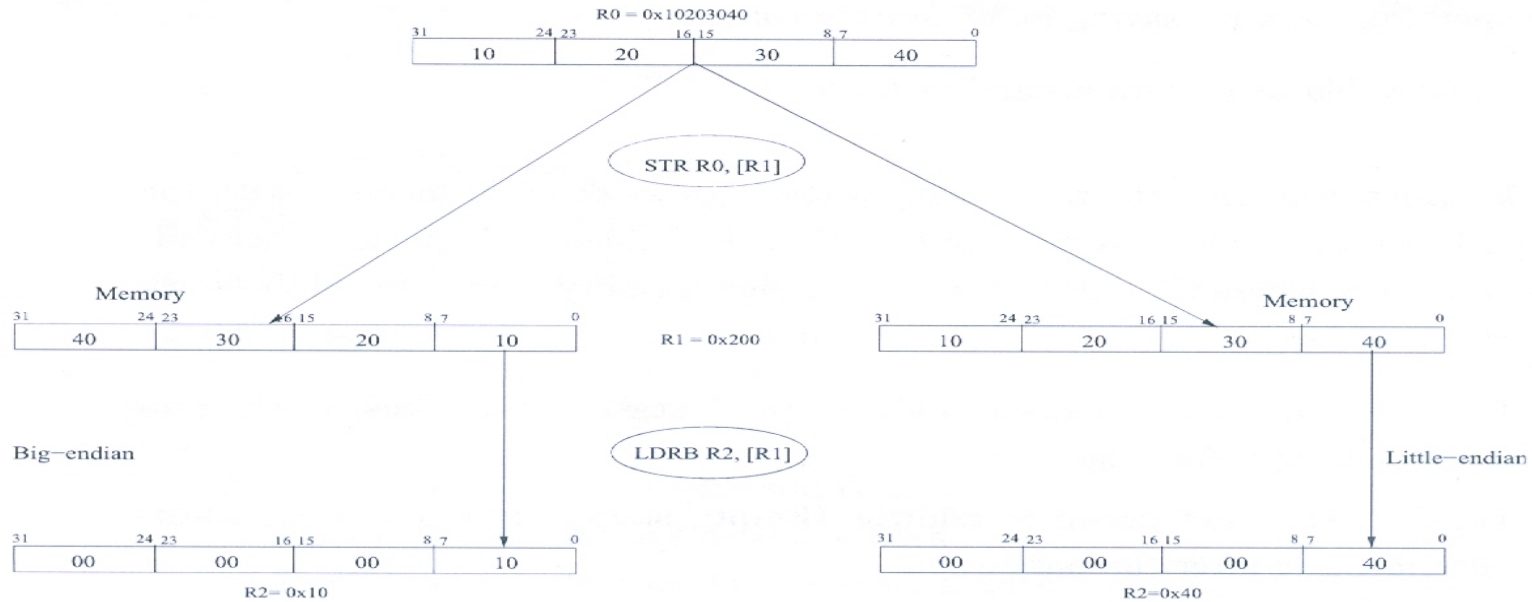
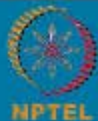
**LDR R0, [R1, #4]!; R0 = Memory[R1 + 4]
 R1 = R1 + 4**

**LDR R0, [R1], #16; R0 = Memory[R1]
 R1 = R1 + 16**

Various Load/Store Instructions

LDR	Load word	STR	Store word
LDRH	Load half word	STRH	Store half word
LDRSH	Load signed half word	STRSH	Store signed half word
LDRB	Load byte	STRB	Store byte
LDRSB	Load signed byte	STRSB	Store signed byte

Little-endian vs. Big-endian



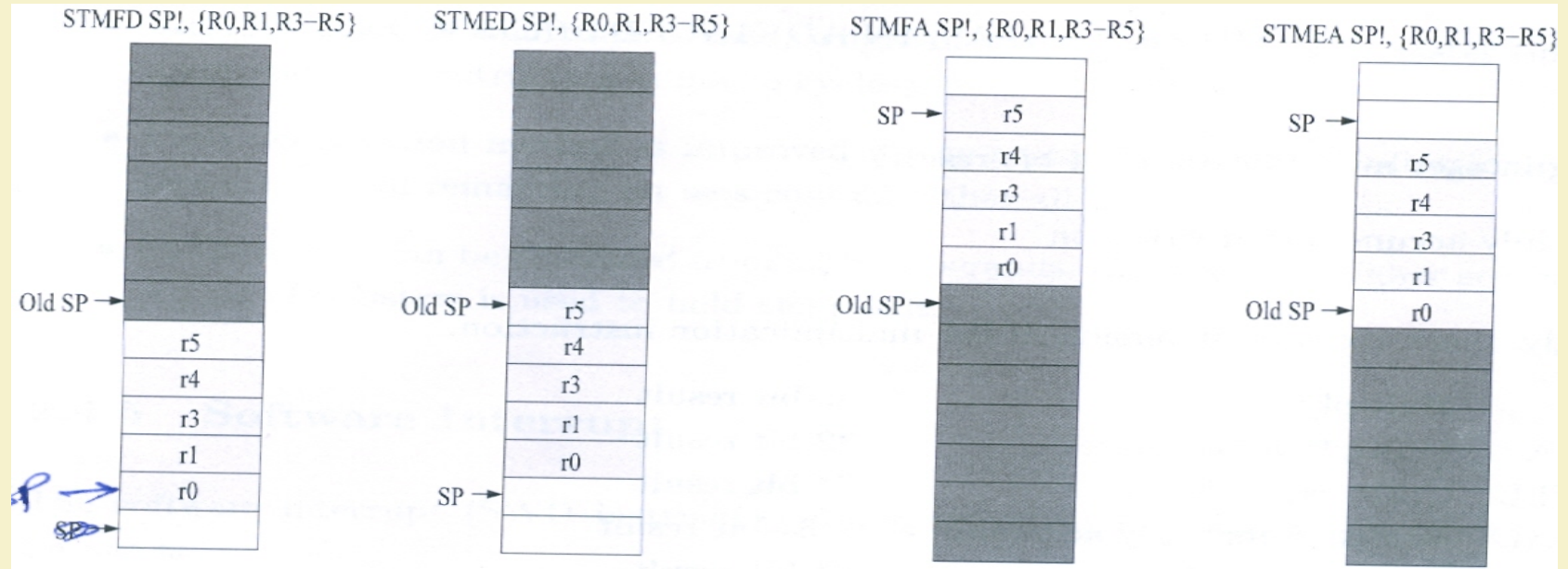
Block Data Transfer

- Load and Store multiple instructions (LDM/STM) allow between 1 and 16 registers to be transferred to or from memory
- Transferred registers can be
 - Any subset of current bank of registers (default)
 - Any subset of user mode bank of registers when in privileged mode
- Can use base register, auto-increment and decrement
- Lowest register number is always transferred to/from lowest memory location accessed
- Can be utilized in
 - Implementing stack
 - Moving large blocks of data around memory

Stack Implementation

- **Descending or ascending stack**
- **Full (stack pointer points to the last occupied address) or empty (stack pointer points to the next available address)**
- **Various instructions:**
 - **STMFD/LDMFD: Full descending stack**
 - **STMFA/LDMFA: Full ascending stack**
 - **STMED/LDMED: Empty descending stack**
 - **STMEA/LDMEA: Empty ascending stack**

Stack Example



Moving Large Data Block

- **Instructions –**
 - **STMIA/LDMIA: Increment after**
 - **STMIB/LDMIB: Increment before**
 - **STMDA/LDMA: Decrement after**
 - **STMDB/LDMDB: Decrement before**
- **Example –**
 - ; R12 points to start of source data**
 - ; R14 points to the end of source data**
 - ; R13 points to the start of the destination data**
 - Loop LDMIA R12!, {R0-R11}; load 48 bytes**
 - STMIA R13!, {R0-R11}; and store them**
 - CMP R12, R14; check for the end**
 - BNE Loop; and loop until done**

Multiplication Instruction

- Several versions
 - Integer multiplication (32-bit result)
 - Long integer multiplication (64-bit result)
 - Multiply accumulate instruction
- Instructions
 - MUL – 32 bit multiply
 - MULA - 32-bit multiply accumulate
 - UMULL – 64-bit unsigned multiply
 - UMLAL – 64-bit unsigned multiply accumulate
 - SMULL – 64-bit signed multiply
 - SMLAL – 64-bit signed multiply accumulate
- Example
 - MUL R0, R1, R2; $R0 = R1 * R2$
 - MULA R0, R1, R2, R3; $R0 = R1 * R2 + R3$

Multiplication Instruction (Contd.)

- Destination and source cannot be the same register
- PC (R15) cannot be used for multiplication
- Uses Booth's algorithm
- For each pair of bits, it takes 1 cycle
- One more cycle needed to start the instruction
- Multiplication continues till source register has some 1's left. Otherwise it *early-terminates*
- To multiply 18 by -1, if 18 is in source, it takes 4 cycles, whereas, if -1 is the source, it needs 17 cycles

Software Interrupt

- Forces CPU to supervisor mode
- Instruction format: SWI # n
- Causes an exception trap to the SWI hardware vector, exception handler is called
- Exception handler analyzes the value of n to determine the action
- Processor completely ignores n
- Used to implement *system calls*
- Value of n is 24-bit, allowing 2^{24} different system calls

Conditional Execution

- ARM allows all instructions to be executed conditionally
- Most significant 4-bits of each instruction are reserved to hold 16 condition codes
- Instruction is executed only if the condition set is met by the flags in CPSR
- Example:
 - EQADD R0, R1, R2; $R0 = R1 + R2$ only if zero flag is set