ARM Instruction Set

Data Transfer Instructions & Control Flow Instructions

Dr. Munesh Singh

Indian Institute of Information Technology Design and Manufacturing
Kancheepuram, Chennai, Tamil Nadu

January 19, 2020

Data transfer instructions

- ARM instruction set support three types of data transfers:
  1. Single register loads and stores
     - Flexible, supports byte, half-word and word transfers
  2. Multiple register loads and stores
     - Less flexible, multiple words, higher transfer rate
  3. Single register-memory swap
     - Mainly for system use (for implementing locks)

- All ARM data transfer instructions use register indirect addressing.
  - Before any data transfer, some register must be initialized with a memory address
    ADRL r1, Table; r1=memory address of Table
  - Example:
    LDR r0, [r1]; r0=mem[r1]
    STR r0, [r1]; mem[r1]= r0

Single register loads and stores

- The simplest form uses register indirect without any offset:
  LDR r0, [r1]; r0=mem[r1]
  STR r0, [r1]; mem[r1]= r0
- An alternative form uses register indirect with offset (limited to 4 kbytes):
  LDR r0, [r1,#4]; r0=mem[r1+4]
  STR r0, [r1,#12]; mem[r1+12]= r0
- We can also use auto-indexing in addition:
  LDR r0, [r1,#4]!; r0=mem[r1+4], r1=r1+4
  STR r0, [r1,#12]!; mem[r1+12]= r0, r1=r1+4
- We can use post indexing:
  LDR r0, [r1],#4]!; r0=mem[r1], r1=r1+4
  STR r0, [r1],#12]!; mem[r1]= r0, r1=r1+12

Single register loads and stores

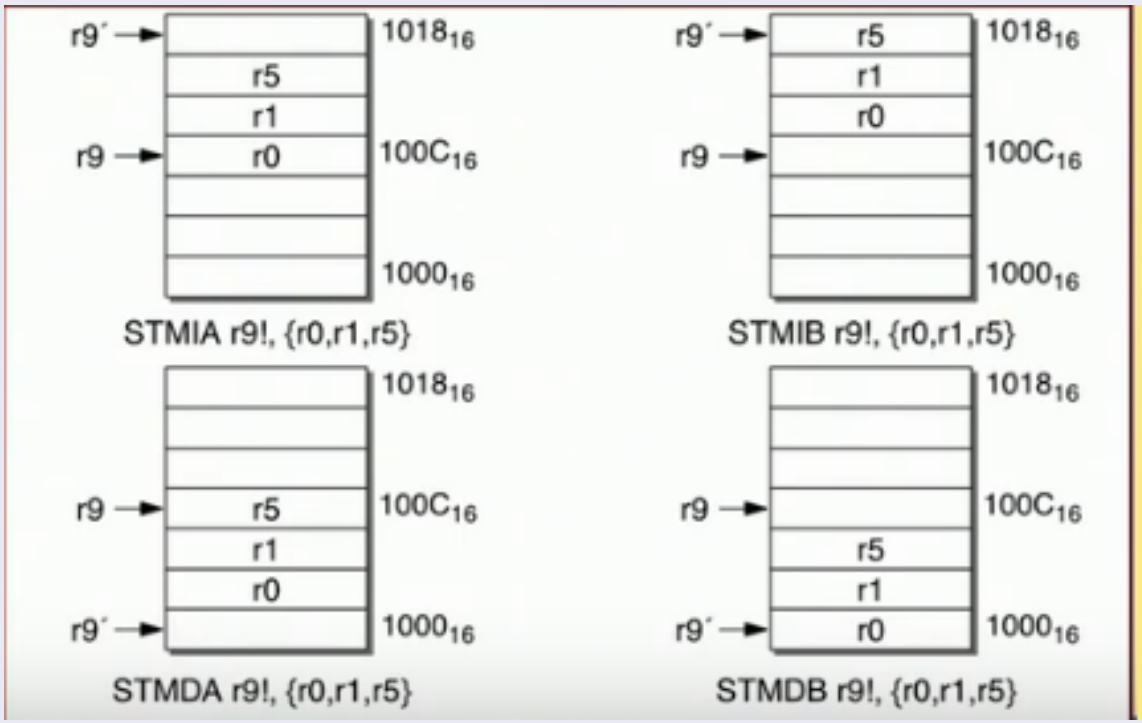
- We can specify a byte or half-word to be transferred:
  LDRB r0, [r1]; r0=mem8[r1]
  STRB r0, [r1]; mem8[r1]= r0
  LDRSH r0, [r1]; r0=mem16[r1]
  STRSH r0, [r1]; mem16[r1]= r0

Multiple register loads and stores

- ARM support instructions that transfer between several registers and memory.
  LDMIA r1, {r3,r5,r6};
  r3=mem[r1],
  r5=mem[r1+4],
  r6=mem[r1+8]
- For LDMIB, the addresses will be r1+4,r1+8 and r1+12.
- The list of the destination registers may contain any or all of r0 to r15.
- Block copy addressing
  - Supported with addresses that can increment(I) or decrement(D), before(B) or after(A) each transfer.

Multiple register loads and stores

- Examples of addressing modes in multiple-register transfer:



LDMIA, STMIA
- Increment after

LDMIB and STMIB
- Increment before

LDMDA, STMDA
- Decrement after

LDMDB, STMDB
- Decrement before

- Point to note:
  - ARM does not support any hardware stack.
  - Software stack can be implemented using the LDM and STM family of instructions.

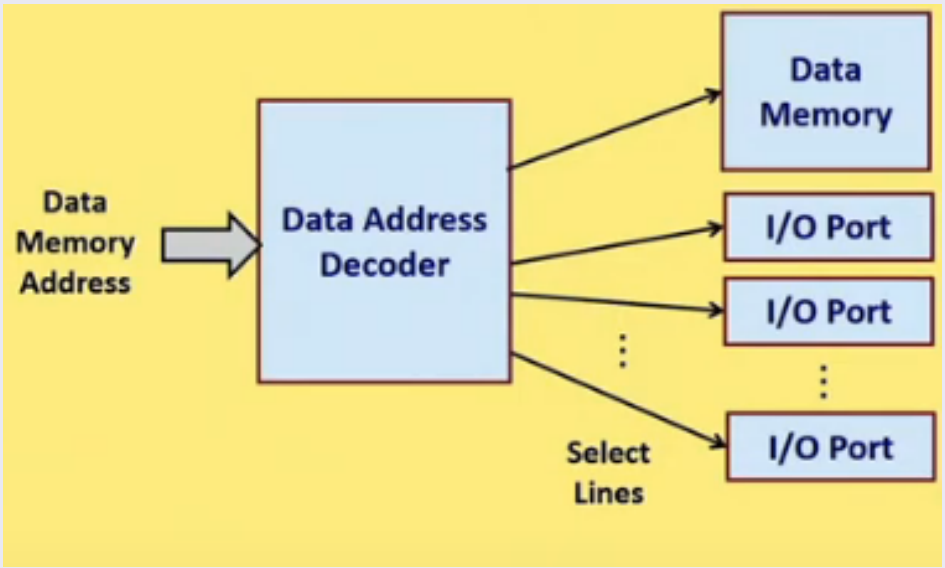
An Example

- Copy a block of memory (128 bytes aligned)
  - r9: address of the source
  - r10: address of the destination
  - r11: end address of the source

```
Loop:   LDMIA   r9!,{r0-r7}
        STMIA   r10!,{r0-r7}
        CMP     r9,r11
        BNE     Loop
```

Memory Mapped I/O in ARM

- No separate instructions for input/output.
- The I/O ports are treated as data memory locations.
  - Each with a unique (memory) address
- Data input is done using the LDR instruction.
- Data output is done with the STR instruction.



Control Flow Instructions

- These instructions change the order of instruction execution.
  - Normal flow is sequential execution, where PC is incremented by 4 after instruction
- Types of conditional flow instructions:
  - Unconditional branch
  - Conditional branch
  - Branch and Link
  - Conditional execution

Control Flow Instructions

Unconditional branch instruction:

```
            B       Target
            ...
            ...
Target      ...
```

Conditional branch instruction:

```
            MOV     r2,#0
LOOP        ...
            ...
            ADD     r2,r2,#1
            CMP     r2,#20
            BNE     LOOP
            ...
```

Control Flow Instructions

- Branch conditions that are supported:
- B, BAL Unconditional branch
- BEQ,BNE Equal or not equal to zero
- BPL, PMI Result positive or negative
- BCC,BCS Carry set or clear
- BVC, BVS Overflow set or clear
- BGT,BGE Greater than, greater or equal
- BLT,BLE Less than, less or equal

Branch and link instruction

- Used for calling subroutines in ARM
- The return address is saved in register r14 (called link register)
- To return from the subroutine, we have to jump back to the address stored in r14.

```
            BL    MYSUB        ; Branch to subroutine
            ...                ; Return here
            ...
MYSUB       ...                ; Subroutine starts here
            ...
            MOV   pc,r14       ; Return
```

Nested subroutine calls cannot be used in this way.

- We can use software stack to save/restore the return address and registers.
- An example showing nested subroutine calls and return.

```
            BL      MYSUB1
            ...

MYSUB1      STMFD   r13!,{r0-r2,r14}
            BL      MYSUB2
            ...
            LDMFD   r13!,{r0-r2,pc}

MYSUB2      ...
            ...
            MOV     pc,r14
```

Conditional Execustion

- A unique feature of the ARM instruction set.
- All instructions can be made conditional, i.e will get executed only when a specified condition is true.
- Help in removing many short branch instructions (improves performance and code density)
- An example: if (r2!=10) r5=r5+10-r3

ARM Instruction

```
CMP r2, #10
ADDNE r5,r5,r2
SUBNE r5,r5,r3
```

Thumb Instruction

```
CMP r2, #10
BEQ SKIP
ADD r5,r5,r2
SUB r5,r5,r3
SKIP.....
```

- Various instruction postfix supported for conditional execution:

Postfix	Condition	Postfix	Condition
CS	Carry set	CC	Carry clear
EQ	Equal (zero set)	NE	Not equal (zero clear)
VS	Overflow set	VC	Overflow clear
GT	Greater than	LT	Less than
GE	Greater than or equal	LE	Less than or equal
PL	Plus (positive)	MI	Minus (negative)
HI	Higher than	LO	Lower than (i.e. CC)
HS	Higher or same (i.e. CS)	LS	Lower or same

- Another example:
  if ((r1==r3) && (r5==r6)) r7=r7+10

- Another example:
  if ((r1==r3) && (r5==r6)) r7=r7+10

ARM Instruction

```
CMP r1, r3
CMPEQ r5,r6
ADDEQ r7,r7,#10
```

Thumb Instruction

```
CMP r1, r3
BNE SKIP
CMP r5,r6
BNE SKIP
ADD r7,r7,#10
SKIP.....
```

Supervisor calls

- The software interrupt instruction (SWI) is used to enter Supervisor Mode, usually to request a particular supervisor function (e.g. input or output).
- The CPSR is saved into the Supervisory Mode SPSR, and execution branches to the SWI vector.
- The SWI handler read the opcode to extract the SWI function number.

Thank You