DATE: 24042020

LOAD STORE INSTRUCTION IN ARM

These instructions are used for moving a single data element into a register from memory or from a memory location to a register.

These instructions can transfer : a 32 bit data, 16 bit data (Halfword) and (8 bit) data
H - 16bit B - 8bit

LOAD MEMORY $\xrightarrow{\text{data}}$ REGISTER
STORE MEMORY $\xleftarrow[\text{data}]{}$ RGISTER

Load Register

1 LDR Rd, Address ; Rd ← mem Address

① This instruction means load the contents from memory to Register
② The size of register and memory are 32-bits in size

store register

2. STR Rd, Address ; Rd → mem Address

(a) This instruction means Store the contents from Register to Memory

(b) The size of register and memory are 32-bits size.

DATE: 24042020

LOAD STORE INSTRUCTION IN ARM

These instructions are used for moving a single data element into a register from memory or from a memory location to a register.

These instructions can transfer a 32 bit data, 16 bit data (Half word) and (8 bit) data
H - 16bit B - 8bit

LOAD MEMORY $\xrightarrow{\text{data}}$ REGISTER
STORE MEMORY $\xleftarrow{\text{data}}$ REGISTER

Load Register

1. LDR Rd, Address ; Rd ← mem Address

① This instruction means load the contents from memory to Register
② The size of register and memory are 32-bits in size

store register

2. STR Rd, Address ; Rd → mem Address

(a) This instruction means Store the contents from Register to Memory

(b) The size of register and memory are 32-bits size.

DATE: ☐☐☐☐☐☐

Load Store for Byte operations

LDR (B) Rd, Address ; Rd ← mem
data size (8 bits)

(a) This instruction means load the contents of memory to register.

(b) The data size is 8 bits

STR B Rd, Address; Rd → memory
8 bits

(a) This instruction means store the contents from register to memory

(b) The data size is 8 bits.

DATE: ☐☐☐☐☐☐

Load and store for Half word (16bit data)

LDRH Rd, Address ; Rd ← memory (16 bits)

STRH Rd, Address ; Rd → memory

Similarly instructions for sign extended data,

LDRSB Rd, Address ; Rd ← mem Sign extended Rd ← mem

LDRSH Rd, Address ; Rd ← memory Sign half word to a register.

DATE: [0][4][0][5][2][0][2][3]

Software Interrupt Instructions (SWI)

→ These instructions provides a mechanism for application to call operating system routines

→ It is like int 21h, int 10h in 8086 (raw DOS instructions)

Syntax : SWI {<cond>} SWI_number

Here on execution of this interrupt-

SPSR_SVC = CPSR copy CPSR to SPSR_SVC to allow OS routine to be called in privileged mode

SVC → Supervisor calls.

PC = Vector + 0x8 sets PC to 0 offset- 0x8 in vector table.

CPSR mode = SVC

CPSR I = 1 (mask IRQ Interrupt)

→ It is a 32-bit instruction

lr_SVC = address of instruction following SWI

DATE: ☐☐☐☐☐☐

Example

SWI is invoked with SWI_number 0x123456 used by ARM toolkit as a debugging SWI in user mode

Before: CPSR = nzcvqift_USER

PC = 0x00008000

Lr = 0x003fffff

lr = r14, r0 = 0x12

0x00008000 SWI 0x123456

After Execution

CPSR = nzcvqift_SVC

SPSR = nzcvqift_USER

PC = 0x00000008

lr = 0x00008004, r0 = 0x12

Since SWI instructions are used to call OS routines, it needs parameter passing mechanism using registers.

DATE: ☐☐☐☐☐☐☐☐

Program Status Register Instructions (MRS, MSR)

There are two instructions to control status register (CPSR, SPSR) MRS, MSR

CPSR - Current Program Status Register

SPSR - Saved Program Status Registers.

MRS → Instruction transfers contents of CPSR/SPSR to a register

CPSR/SPSR → contents → Registers general purpose registers

Syntax:-

MRS{<cond>} Rd, <CPSR/SPSR>;

Status registers to general purpose registers.

R → is destination

S → is status source

DATE: ☐☐☐☐☐☐

MSR {<Cond>} <CPSR | SPSR> <fields>, Rm

R is source, S is destination status

MSR {<Cond>} <CPSR | SPSR>_<field>, #immediate

Here <field> can be a combination of control(C) extension (X) status (S) and flags (S) of PSR byte fields.

In user mode we can read all CPSR bits and can only write condition flags field

If IRQ is disabled now enable IRQ is

~~CSP~~ (CPSR = nzcvq/Ift_SVC)

Examples:

```
MRS r1, CPSR      ; r1 = F*******3
BIC r1, r1 #0x80;
MSR CPSR_C, r1
```

After CPSR_C, ~~PC~~ = nCzVq/Ift_SVC

DATE: □□ □□ □□□□

Here MRS copies CPSR to r1,
BIC clear bit-7
r1 is copied back to CPSR to enable IRQ (i=0)

Note:- In SVC mode,
In User mode we can read all CPSR bits, but can update condition flag field f.