18CS44: IAT2 solution

Faculty: Dr. Imtiyaz Ahmed & Prof.Preethi A

1. ***Stack operations in ARM***

- The ARM architecture uses the load-store multiple instructions to carry out stack operations.
- The pop operation (removing data from a stack) uses a load multiple instruction; similarly, the push operation (placing data onto the stack) uses a store multiple instruction.
- When you use a **full stack (F)**, the stack pointer sp points to an address that is the last used or full location.
- In contrast, if you use an **empty stack (E)** the sp points to an address that is the first unused or empty location.
- A stack is either ascending (A) or descending (D). Ascending stacks grow towards higher memory addresses; in contrast, descending stacks grow towards lower memory addresses.
- Addressing modes for stack operation

Addressing mode	Description
FA	full ascending
FD	full descending
EA	empty ascending
ED	empty descending

- The LDMFD and STMFD instructions provide the pop and push functions, respectively.
- Example1: With full descending

```
PRE   r1 = 0x00000002
      r4 = 0x00000003
      sp = 0x00080014

      STMFD   sp!, {r1,r4}

POST  r1 = 0x00000002
      r4 = 0x00000003
      sp = 0x0008000c
```

PRE	Address	Data
	0x80018	0x00000001
sp →	0x80014	0x00000002
	0x80010	*Empty*
	0x8000c	*Empty*

POST	Address	Data
	0x80018	0x00000001
	0x80014	0x00000002
	0x80010	0x00000003
sp →	0x8000c	0x00000002

Figure: STMFD instruction full stack push operation.

Example 2: With empty descending

```
PRE     r1 = 0x00000002
        r4 = 0x00000003
        sp = 0x00080010

        STMED   sp!, {r1,r4}

POST    r1 = 0x00000002
        r4 = 0x00000003
        sp = 0x00080008
```

PRE

Address	Data
0x80018	0x00000001
0x80014	0x00000002
0x80010	*Empty*
0x8000c	*Empty*
0x80008	*Empty*

sp →

POST

Address	Data
0x80018	0x00000001
0x80014	0x00000002
0x80010	0x00000003
0x8000c	0x00000002
0x80008	*Empty*

sp →

Figure: STMED instruction empty stack push operation.

Addressing methods for stack operations.

Addressing mode	Description	Pop	= LDM	Push	= STM
FA	full ascending	LDMFA	LDMDA	STMFA	STMIB
FD	full descending	LDMFD	LDMIA	STMFD	STMDB
EA	empty ascending	LDMEA	LDMDB	STMEA	STMIA
ED	empty descending	LDMED	LDMIB	STMED	STMDA

2. **syntax of following multiply instructions with clear examples;**
i. MLA, ii. SMLAL, iii.UMULL

The multiply instructions multiply the contents of a pair of registers and, depending upon the instruction, accumulate the results in with another register. The long multiplies accumulate onto a pair of registers representing a 64-bit value. The final result is placed in a destination register or a pair of registers.

long multiply instructions (SMLAL, SMULL, UMLAL, and UMULL) produce a 64-bit result. The result is too large to fit a single 32-bit register so the result is placed in two registers labeled *RdLo* and *RdHi*. *RdLo* holds the lower 32 bits of the 64-bit result, and *RdHi* holds the higher 32 bits of the 64-bit result. Example 3.12 shows an example of a long unsigned multiply instruction

```
Syntax: MLA{<cond>}{S} Rd, Rm, Rs, Rn
        MUL{<cond>}{S} Rd, Rm, Rs
```

MLA	multiply and accumulate	$Rd = (Rm*Rs) + Rn$
MUL	multiply	$Rd = Rm*Rs$

```
Syntax: <instruction>{<cond>}{S} RdLo, RdHi, Rm, Rs
```

SMLAL	signed multiply accumulate long	$[RdHi, RdLo] = [RdHi, RdLo] + (Rm*Rs)$
SMULL	signed multiply long	$[RdHi, RdLo] = Rm*Rs$
UMLAL	unsigned multiply accumulate long	$[RdHi, RdLo] = [RdHi, RdLo] + (Rm*Rs)$
UMULL	unsigned multiply long	$[RdHi, RdLo] = Rm*Rs$

```
PRE    r0 = 0x00000000
       r1 = 0x00000000
       r2 = 0xf0000002
       r3 = 0x00000002

       UMULL   r0, r1, r2, r3   ; [r1,r0] = r2*r3

POST   r0 = 0xe0000004 ; = RdLo
       r1 = 0x00000001 ; = RdHi
```

3. **syntax of BX instruction. With example pseudocode illustrate offset calculation of forward and backward jump**

The **BX instruction** causes a **branch to the address** contained **in Rm** and **exchanges the instruction set**, if required:

If bit[0] of Rm is 0, the processor changes to, or remains in, ARM state.
If bit[0] of Rm is 1, the processor changes to, or remains in, Thumb state.

```
Syntax: B{<cond>} label
        BL{<cond>} label
        BX{<cond>} Rm
        BLX{<cond>} label | Rm
```

B	branch	*pc* = *label*
BL	branch with link	*pc* = *label* *lr* = address of the next instruction after the BL
BX	branch exchange	*pc* = *Rm* & 0xfffffffe, *T* = *Rm* & 1
BLX	branch exchange with link	*pc* = *label*, *T* = 1 *pc* = *Rm* & 0xfffffffe, *T* = *Rm* & 1 *lr* = address of the next instruction after the BLX

- **Change of execution flow** forces the **program counter pc to** point to a **new address**

- **Offset is Calculated as**
- **label (assigned to address of target instruction)-PC (address of next instruction)=+/- offset**
- **+ offset=forward jump and – offset backward jump**

```
        B     forward
        ADD   r1, r2, #4
        ADD   r0, r6, #2
        ADD   r3, r7, #4
forward
        SUB   r1, r2, #4
```

4. **i. 16 bit load /store, ii. Signed 16 bit load /store, iii.Signed 8 bit load /store And Indexing methods of load /store instructions.**

LDR{<cond>} H Rd, addressing^2

STR{<cond>} H Rd, addressing^2

LDR{<cond>}SH Rd, addressing^2

STR{<cond>}H Rd, addressing^2

LDR{<cond>}SB Rd, addressing^2

STR{<cond>}B Rd, addressing^2

- **No STRSB or STRSH instructions since STRH stores both a signed and unsigned halfword;**
- **Similarly STRB stores signed and unsigned bytes.**

Single-register load-store addressing, halfword, signed halfword, signed byte, and doubleword.

Addressing^2 mode and index method	Addressing^2 syntax
Preindex immediate offset	[Rn, #+/-offset_8]
Preindex register offset	[Rn, +/-Rm]
Preindex writeback immediate offset	[Rn, #+/-offset_8]!
Preindex writeback register offset	[Rn, +/-Rm]!
Immediate postindexed	[Rn], #+/-offset_8
Register postindexed	[Rn], +/-Rm

Variations of STRH instructions.

	Instruction	Result	*r1* +=
Preindex with writeback	STRH r0,[r1,#0x4]!	mem16[r1+0x4]=r0	0x4
	STRH r0,[r1,r2]!	mem16[r1+r2]=r0	r2
Preindex	STRH r0,[r1,#0x4]	mem16[r1+0x4]=r0	*not updated*
	STRH r0,[r1,r2]	mem16[r1+r2]=r0	*not updated*
Postindex	STRH r0,[r1],#0x4	mem16[r1]=r0	0x4
	STRH r0,[r1],r2	mem16[r1]=r0	r2

5. **instructions to operate on Program Status Register**

- The ARM instruction set provides two instructions to directly control a program status register (psr).
- The MRS instruction transfers the contents of either the cpsr or spsr to general purpose register.
- The MSR instruction transfers the contents of a general purpose register to cpsr or spsr.
- Together these instructions are used to read and write the cpsr and spsr.

Syntax: MRS {<cond>} Rd <cpsr |spsr>
MSR {<cond>} <cpsr|spsr} _<fields>,Rm
MSR {<cond>} <cpsr|spsr} _<fields>, #immediate

- The table shows the program status register instructions

MRS	copy program status register to a general-purpose register	*Rd = psr*
MSR	move a general-purpose register to a program status register	*psr[field] = Rm*
MSR	move an immediate value to a program status register	*psr[field] = immediate*

6. i.SWP, ii. SWI

Swap Ins:

- It is a special case of a load-store instruction.
- It swaps the contents of memory with the contents of a register.
- This instruction is an atomic operation—{it reads and writes a location in the same bus operation, preventing any other instruction from reading or writing to that location until it completes}
- Swap instruction cannot be interrupted by any other instruction or any other bus access. {"holds the bus" until the transaction is complete}

```
Syntax: SWP{B}{<cond>} Rd,Rm,[Rn]
```

SWP	swap a word between memory and a register	*tmp = mem32[Rn]* *mem32[Rn] = Rm* *Rd = tmp*
SWPB	swap a byte between memory and a register	*tmp = mem8[Rn]* *mem8[Rn] = Rm* *Rd = tmp*

```
PRE     mem32[0x9000] = 0x12345678
        r0 = 0x00000000
        r1 = 0x11112222
        r2 = 0x00009000

        SWP     r0, r1, [r2]

POST    mem32[0x9000] = 0x11112222
        r0 = 0x12345678
        r1 = 0x11112222
        r2 = 0x00009000
```

- A software interrupt instruction (SWI) causes a software interrupt exception, which provides a mechanism for applications to call operating system routines.

Syntax: SWI {<cond>} SWI_number

SWI	software interrupt	*lr_svc* = address of instruction following the SWI *spsr_svc* = *cpsr* *pc* = vectors + 0x8 *cpsr* mode = *SVC* *cpsr I* = 1 (mask IRQ interrupts)

- When the processor executes an SWI instruction, it sets the program counter pc to the offset 0xB in the vector table.
- The instruction also forces the processor mode to SVC, which allows an operating system routine to be called in a privileged mode.
- Each SWI instruction has an associated SWI number, which is used to represent a particular function call or feature.
- The example below shows an SWI call with SWI number 0x123456, used by ARM toolkits as a debugging SWI.

```
PRE    cpsr = nzcVqift_USER
       pc = 0x00008000
       lr = 0x003fffff; lr = r14
       r0 = 0x12

     0x00008000   SWI     0x123456

POST   cpsr = nzcVqIft_SVC
       spsr = nzcVqift_USER
       pc = 0x00000008
       lr = 0x00008004
       r0 = 0x12
```

- Since SWI instructions are used to call operating system routines, it is required some form of parameter passing.
- This achieved by using registers. In the above example, register r0 is used to pass parameter 0x12. The return values are also passed back via register.

7. Embedded System. Differentiate between General Computing and Embedded Computing systems

An embedded system is a combination of 3 types of components: a. Hardware b. Software c. Mechanical Components and it is supposed to do one specific task only.

Example 1: Washing Machine

- A washing machine from an embedded systems point of view has: a. Hardware: Buttons, Display & buzzer, electronic circuitry. b. Software: It has a chip on the circuit that holds the software which drives controls & monitors the various operations possible. c. Mechanical Components: the internals of a washing machine which actually wash the clothes control the input and output of water, the chassis itself.

Example 2: Air Conditioner

- An Air Conditioner from an embedded systems point of view has: a. Hardware: Remote, Display & buzzer, Infrared Sensors, electronic circuitry. b. Software: It has a chip on the circuit that holds the software which drives controls & monitors the various operations possible. The software monitors the external temperature through the sensors and then releases the coolant or suppresses it. c. Mechanical Components: the internals of an air conditioner the motor, the chassis, the outlet, etc An embedded system is designed to do a specific job only.

Criteria	General Purpose Computer	Embedded system
Contents	It is combination of generic hardware and a general purpose OS for executing a variety of applications.	It is combination of special purpose hardware and embedded OS for executing specific set of applications
Operating System	It contains general purpose operating system	It may or may not contain operating system.
Alterations	Applications are alterable by the user.	Applications are non-alterable by the user.
Key factor	Performance" is key factor.	Application specific requirements are key factors.
Power Consumption	More	Less
Response Time	Not Critical	Critical for some applications

8. **any 3 purposes of Embedded System in detail**

Data Collection/Storage/Representation
Data communication
Data signal processing
Monitoring
Control
Application specific user interface

1. Data Collection/Storage/Representation
   - Embedded system designed for the purpose of data collection performs acquisition of data from the external world.
   - Data collection is usually done for storage, analysis, manipulation and transmission.

- Data can be analog or digital.
- Embedded systems with analog data capturing techniques collect data directly in the form of analog signal whereas embedded systems with digital data collection mechanism converts the analog signal to the digital signal using analog to digital converters.
- If the data is digital it can be directly captured by digital embedded system.
- A digital camera is a typical example of an embedded System with data collection/storage/representation of data.
- Images are captured and the captured image may be stored within the memory of the camera. The captured image can also be presented to the user through a graphic LCD unit.

2. Data communication
    - Embedded data communication systems are deployed inapplications from complex satellite communication to simple home networking systems.
    - The transmission of data is achieved either by a wire-lin medium or by a wire-less medium. Data can either be transmitted by analog means or by digital means.
    - Wireless modules-Bluetooth, Wi-Fi.
    - Wire-line modules-USB, TCP/IP.
    - Network hubs, routers, switches are examples of dedicated data transmission embedded systems.
3. Data signal processing
    - Embedded systems with signal processing functionalities are employed in applications demanding signal processing like speech coding, audio video codec, transmission applications etc.
    - A digital hearing aid is a typical example of an embedded system employing data processing. Digital hearing aid improves the hearing capacity of hearing impaired person.
4. Monitoring
    - All embedded products coming under the medical domain are with monitoring functions. Electro cardiogram machine is intended to do the monitoring of the heartbeat of a patient but it cannot impose control over the heartbeat.
    - Other examples with monitoring function are digital CRO, digital multi-meters, and logic analyzers.
5. Control
    - A system with control functionality contains both sensors and actuators Sensors are connected to the input port for capturing the changes in environmental variable and the actuators connected to the output port are controlled according to the changes in the input variable.
    - Air conditioner system used to control the room temperature to a specified limit is a typical example for CONTROL purpose.

6. Application specific user interface
    - Buttons, switches, keypad, lights, bells, display units etc are application specificuser interfaces.
    - Mobile phone is an example of application specific user interface.
    - In mobile phone the user interface is provided through the keypad, system speaker,vibration alert etc.