EE2016 Micropprocessor Lab & Theory, Aug - Nov, 2022

EE Department, IIT, Madras.

Experiment 5: ARM Assembly - Computations in ARM

1 Aim

To (a) learn the architecture of ARM processor (b) learn basics of ARM instruction set, in particular the ARM instructions pertaining to computations (c) go through example programs and (d) write assembly language programs for the given set of (computational) problems

2 Equipments, Hardware Required

The list of equipments, components required are:

1. KEIL 5 IDE for ARM
2. Flashmagic software for programming flash memory
3. ARM7 hardware kit
4. USB to serial converter
5. Serial cross cable

This is purely an experiment based on emulation. (We were forced to run the lab with only emulation based experiments due to ongoing pandemic situation. The hardware details given here is to understand the context).

3 Background Information

You are strongly adviced to go through the online book by Welsh. The material presented here draws heavily from the above book (first 6 chapters, with 6th chapter in depth).

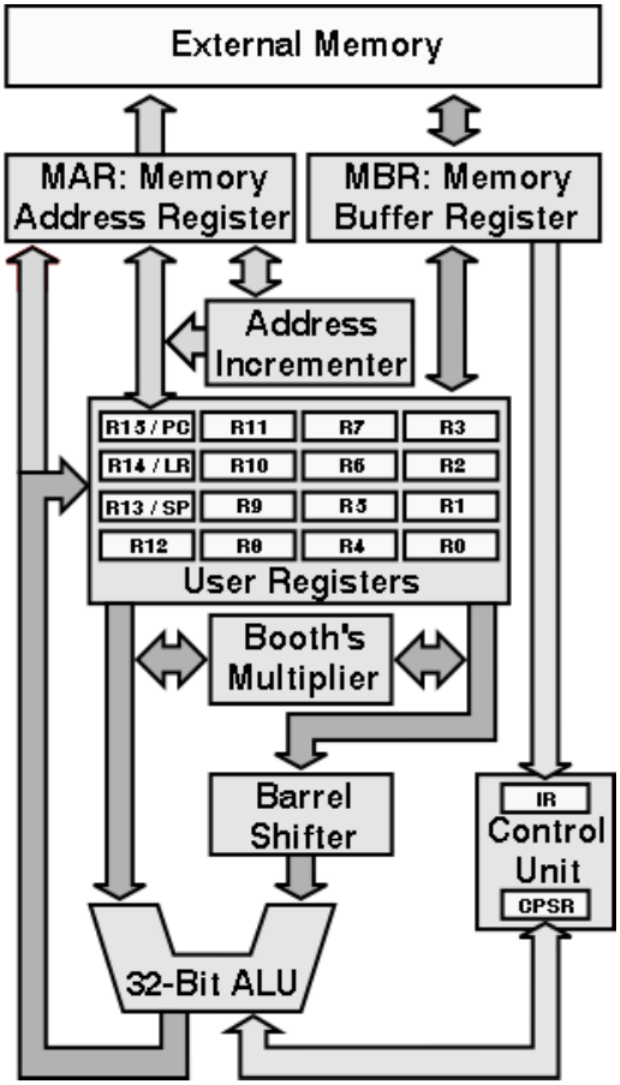
3.1 Review of ARM Architecture

Fig below shows the internal structure of the ARM processor. The ARM is a Reduced Instruction Set Computer (RISC) system and includes the attributes typical to that type of system:

1. A large array of uniform registers.
2. A load/store model of data-processing where operations can only operate on registers and not directly on memory. This requires that all data be loaded into registers before an operation can be preformed, the result can then be used for further processing or stored back into memory.
3. A small number of addressing modes with all load/store addresses begin determined from registers and instruction fields only.
4. A uniform fixed length instruction (32-bit).

In addition to these traditional features of a RISC system the ARM provides a number of additional features:

1. Separate Arithmetic Logic Unit (ALU) and shifter giving additional control over data pro- cessing to maximize execution speed.
2. Auto-increment and Auto-decrement addressing modes to improve the operation of program loops.
3. Conditional execution of instructions to reduce pipeline flushing and thus increase execution speed.



3.1.1 Processor Modes

3.1.2 Registers in ARM Processor

Registers The ARM has a total of 37 registers. These comprise 30 general purpose registers, 6 status registers and a program counter. Figure below illustrates the registers of the ARM. Only fifteen of the general purpose registers are available at any one time depending on the processor mode.

There are a standard set of eight general purpose registers that are always available (R0 – R7) no matter which mode the processor is in. These registers are truly general-purpose, with no special uses being placed on them by the processors' architecture.

A few registers (R8 – reg12) are common to all processor modes with the exception of the fiq mode. This means that to all intent and purpose these are general registers and have no special use. However, when the processor is in the fast interrupt mode these registers and replaced with different set of registers (R8 fiq - R12 fiq). Although the processor does not give any special purpose to these registers they can be used to hold information between fast interrupts. You can consider they to be static registers. The idea is that you can make a fast interrupt even faster by holding information in these registers.

The general purpose registers can be used to handle 8-bit bytes, 16-bit half-words1 , or 32-bit words. When we use a 32-bit register in a byte instruction only the least significant 8 bits are used. In a half-word instruction only the least significant 16 bits are used. Figure 3.3 demonstrates this.

The remaining registers (R13 – R15) are special purpose registers and have very specific roles: R13 is also known as the Stack Pointer, while R14 is known as the Link Register, and R15 is the Program Counter. The "user" (usr) and "System" (sys) modes share the same registers. The exception modes all have their own version of these registers. Making a reference to register R14 will assume you are referring to the register for the current processor mode. If you wish to refer to the user mode version of this register you have refer to the R14 usr register. You may only refer to register from

other modes when the processor is in one of the privileged modes, i.e., any mode other than user mode. There are also one or two status registers depending on which mode the processor is in. The Current Processor Status Register (CPSR) holds information about the current status of the processor (including its current mode). In the exception modes there is an additional Saved Processor Status Register (SPSR) which holds information on the processors state before the system changed into this mode, i.e., the processor status just before an exception.

The stack pointer, SP or R13 Register R13 is used as a stack pointer and is also known as the SP register. Each exception mode has its own version of R13 , which points to a stack dedicated to that exception mode. The stack is typically used to store temporary values. It is normal to store the contents of any registers a function is going to use on the stack on entry to a subroutine. This leaves the register free for use during the function. The routine can then recover the register values from the stack 3.2. REGISTERS 27 on exit from the subroutine. In this way the subroutine can preserve the value of the register and not corrupt the value as would otherwise be the case.

The Link Register, LR or R14 Register R14 is also known as the Link Register or LR. It is used to hold the return address for a subroutine. When a subroutine call is performed via a BL instruction, R14 is set to the address of the next instruction. To return from a subroutine you need to copy the Link Register into the Program Counter. (More in Welsh).

The program counter, PC or R15 Register R15 holds the Program Counter known as the PC. It is used to identify which instruction is to be preformed next. As the PC holds the address of the next instruction it is often referred to as an instruction pointer. The name "program counter" dates back to the times when program instructions where read in off of punched cards, it refers to the card position within a stack of cards. In spite of its name it does not actually count anything!

Reading the program counter When an instruction reads the PC the value returned is the address of the current instruction plus 8 bytes. This is the address of the instruction after the next instruction to be executed2. This way of reading the PC is primarily used for quick, position-independent addressing of nearby instructions and data, including position-independent branching within a program. An exception to this rule occurs when an STR (Store Register) or STM (Store Multiple Registers) instruction stores R15 . The value stored is UNKNOWN and it is best to avoid the use of these instructions that store R15 .

Writing the program counter When an instruction writes to R15 the normal result is that the value written is treated as an instruction address and the system starts to execute the instruction at that address3 .

Current Processor Status Registers: CPSR Rather surprisingly the current processor status register (CPSR) contains the current status of the processor. This includes various condition code flags, interrupt status, processor mode and other status and control information. The exception modes also have a saved processor status register (SPSR), that is used to preserve the value of the CPSR when the associated exception occurs. Because the User and System modes are not exception modes, there is no SPSR available. Figure 3.4 shows the format of the CPSR and the SPSR registers.

The processors' status is split into two distinct parts: the User flags and the Systems Control flags. The upper halfword is accessible in User mode and contains a set of flags which can be used to effect the operation of a program, see section 3.3. The lower halfword contains the System Control information.

Any bit not currently used is reserved for future use and should be zero, and are marked SBZ in the figure. The I and F bits indicate if Interrupts (I) or Fast Interrupts (F) are allowed. The Mode bits indicate which operating mode the processor is in (see 3.1 on page 23). The system flags can only be altered when the processor is in protected mode. User mode programs can not alter the status register except for the condition code flags.

3.1.3 Flags

The upper four bits of the status register contains a set of four flags, collectively known at the condition code. The condition code flags are:

The condition code can be used to control the flow of the program execution. The is often abbreviated to just hcci.

N The Negative (sign) flag takes on the value of the most significant bit of a result. Thus when an operation produces a negative result the negative flag is set and a positive result results in a the negative flag being reset. This assumes the values are in standard two's complement form. If the values are unsigned the negative flag can be ignored or used to identify the value of the most significant bit of the result.

Z The Zero flag is set when an operation produces a zero result. It is reset when an operation produces a non-zero result.

C The Carry flag holds the carry from the most significant bit produced by arithmetic operations or shifts. As with most processors, the carry flag is inverted after a subtraction so that the flag acts as a borrow flag after a subtraction.

V The Overflow flag is set when an arithmetic result is greater than can be represented in a register.

Many instructions can modify the flags, these include comparison, arithmetic, logical and move instructions. Most of the instructions have an S qualifier which instructs the processor to set the condition code flags or not.

3.2 Review of ARM Instruction Sets

Why are a microprocessor's instructions referred to as an instruction set? Because the micropro- cessor designer selects the instruction complement with great care; it must be easy to execute complex operations as a sequence of simple events, each of which is represented by one instruction from a well-designed instruction set.

Assembler often frighten users who are new to programming. Yet taken in isolation, the operations involved in the execution of a single instruction are usually easy to follow. Furthermore, you need not attempt to understand all the instructions at once. As you study each of the programs in these notes you will learn about the specific instructions involved.

Operation Mnemonic	Meaning	Operation Mnemonic	Meaning
ADC	Add with Carry	ORR	Logical OR
ADD	Add	RSB	Reverse Subtract
AND	Logical AND	RSC	Reverse Subtract with Carry
B	Unconditional Branch	SBC	Subtract with Carry
Bcc	Branch on Condition	SMLAL	Mult Accum Signed Long
BIC	Bit Clear	SMULL	Multiply Signed Long
BL	Branch and Link	STM	Store Multiple
CMP	Compare	STR	Store Register (Word)
EOR	Exclusive OR	STRB	Store Register (Byte)
LDM	Load Multiple	SUB	Subtract
LDR	Load Register (Word)	SWI	Software Interrupt
LDRB	Load Register (Byte)	SWP	Swap Word Value
MLA	Multiply Accumulate	SWPB	Swap Byte Value
MOV	Move	TEQ	Test Equivalence
MRS	Load SPSR or CPSR	TST	Test
MSR	Store to SPSR or CPSR	UMLAL	Mult Accum Unsigned Long
MUL	Multiply	UMULL	Multiply Unsigned Long
MVN	Logical NOT		

Table 1 Instruction Mnemonics

Mnemonic	Condition	Mnemonic	Condition
CS	Carry S et	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 (aka CC)
HS	Higher or Same (aka CS)	LS	Lower or Same

Table 2: (Condition Code) Mnemonics

Table 1 lists the instruction mnemonics. This provides a survey of the processors capabilities, and will also be useful when you need a certain kind of operation but are either unsure of the specific mnemonics or not yet familiar with what instructions are available.

See Chapter 4 and Appendix A in Welsh for a detailed description of the individual instructions and chapters 6 through to 12 therein for a discussion on how to use them.

The ARM instruction set can be divided into six broad classes of instruction.

1. Data Movement
2. Arithmetic
3. Memory Access
4. Logical and Bit Manipulation
5. Flow Control
6. System Control / Privileged

Before we look at each of these groups in a little more detail there are a few ideas which belong to all groups worthy of investigation.

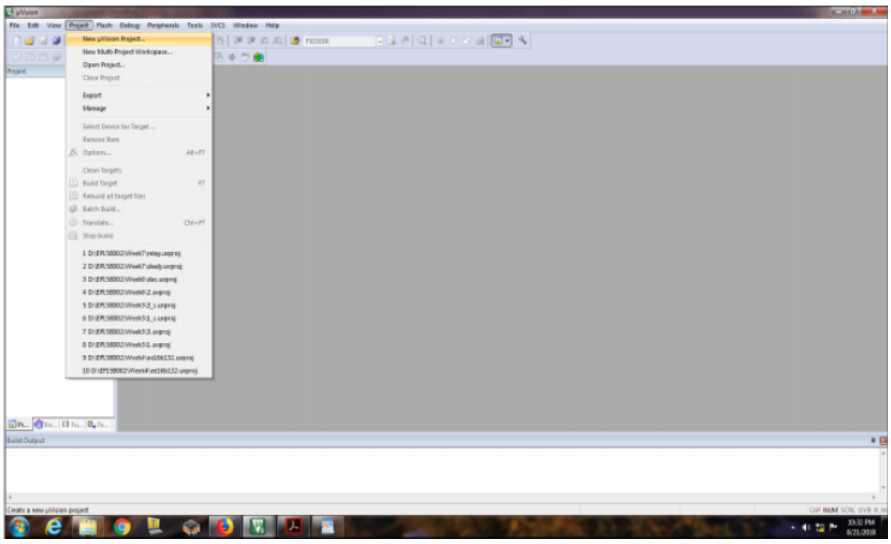
3.3 Overview of KEIL Software

It is very similar to the AVR Studio except that it has an additional feature as explained below

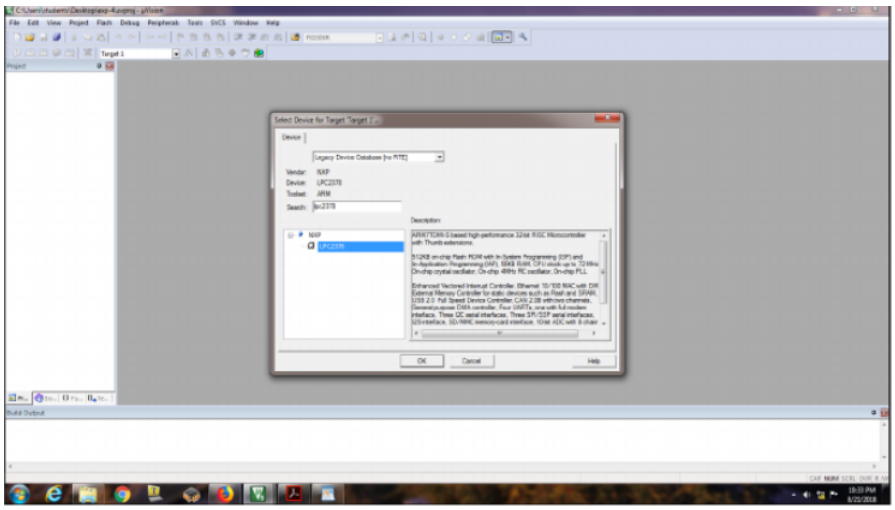
- Keil u Vision is an IDE directed towards code development for multiple platforms like AVR, ARM, CORTEX-M, C166, C251, C51 and 8051 based MCU architectures manufactured by various companies .Whereas Atmel Studio is a Visual Basic and .NET Framework based IDE which only supports AVR and ARM architecture based MCU's only by Atmel.

3.4 Instructions for writing assembly language programs in keil uVision and to execute in the Vi-ARM Kit.

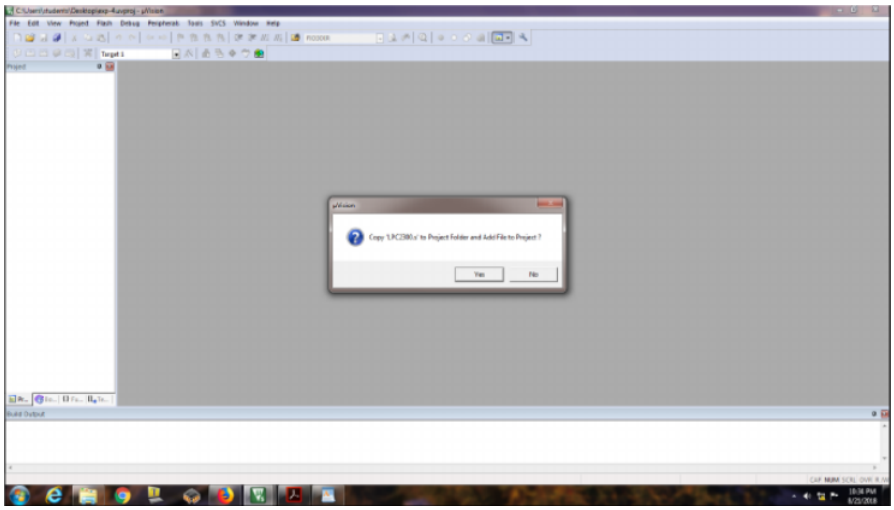
- step1. Open keil uVision
- step2. Click project > New uVision Project



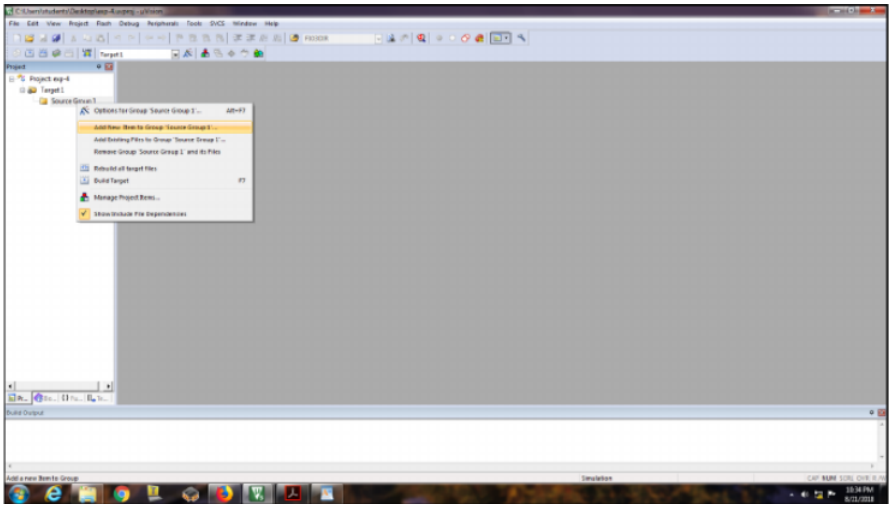
- step3. Select the device LPC2378 under NXP



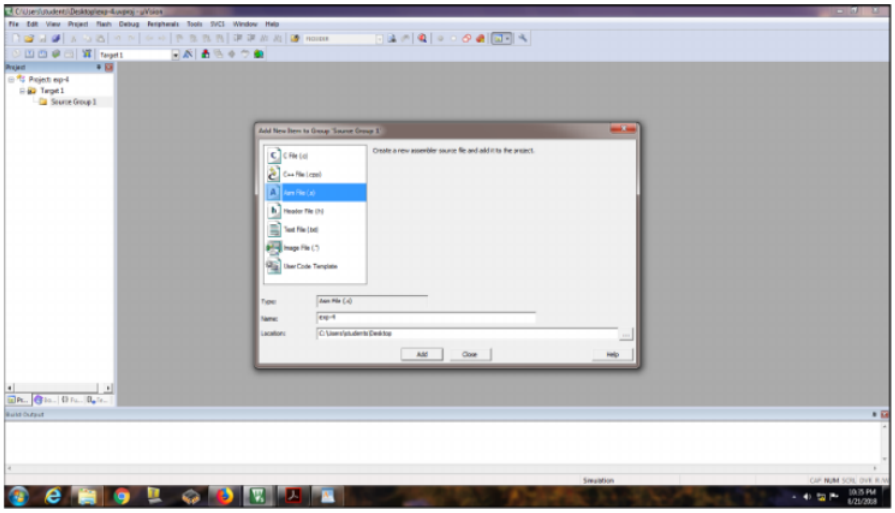
- step4. Copy the Startup LPC23xx.s file. (Choose “NO”)



- step5. Right click Source Group1 under Target in the left side of the keil window. Select “Add new Item to group ”..



- Step6. Select asm(.s) file in the window prompted, give a filename and save.



- step7. Write your program or copy the code from existing program, and save the file

3.5 Example Programs in ARM Assembly Language

Following examples you need to look into and understand them thoroughly (refer Welsh).
(a) 16-bit addition (P.74) (b) bytes disassembly (p.76) and (c) larger of two given numbers (p.77)

4 Demo Program

```
AREA abc,CODE,READONLY ;
LDR R0,NUM1
LDR R1,NUM2
ADD R2,R0,R1
SWI &11
NUM1 DCW &2D3F
align
NUM2 DCW &4C27
END
```

- **AREA abc, CODE, READONLY** : This tells the assembler where the first executable instruction is located and instructs it to assemble a new code(READONLY)
- **SWI &11**: Software interupt -call the operating system [exit()]
- **DCW**: As ARM2378 is 32-bit processor, DCW directive is used to declare a half-word (16-bit)

- **align**: This directive is used to align data item on a 32-bit word boundary (esp. when a 16-bit half-word is read, while accommodating in a 32-bit word).
- **END**: End of program source

5 Tasks: Engineering Problem

Solve the following engineering problems using ARM through assembly programs

1. Compute the factorial of a given number using ARM processor through assembly programming
2. Combine the low four bits of each of the four consecutive bytes beginning at LIST into one 16-bit halfword. The value at LIST goes into the most significant nibble of the result. Store the result in the 32-bit variable RESULT.
3. Given a 32 bit number, identify whether it is an even or odd. (You implementation should not involve division).

6 Procedure

Since it is a simulation experiment, we dont need hardware. It is enough if we have a PC loaded with Keil software.

1. Go through Welsh thoroughly. Do all the home work - meaning start from ARM architecture, go on till example programs. Demo all the example programs in KEIL for yourselves.
2. Write the assembly programs for the above problems (one at a time).
3. Enter the above program in KEIL software, edit and compile / assemble.
4. Run it in the 'debug' mode to see whats happening to the registers.
5. Finally, demonstrate its working, before your TA

7 Results

Show the results to your TA. Send the code TA for evaluation

8 Evaluation Scheme

1. Usually there would an Moodle based online MCQ test
2. Regular lab session marks given by your caretaker TA
3. Report (for this experiment) after submission would be evaluated by TAs. (You need to submit it through the link in moodle created for that purpose, before the deadline given therein).

9 References

The main reference for ARM (remaining part of the course considers only experiments based on ARM), is welsh, which has been posted in moodle. Else you can find it here. http://arantxa.ii.uam.es/~gdrivera/sed/docs/ARMBook.pdf