4th Year 1st Semester

Assistant Professor - ECE, VFSTR, Guntur, AP.

UNIT - IV

CSE

Company

Dr. V.Vijayaraghavan, Assistant Professor-ECE,
VFSTR, Guntur

ARM Programming

Assembly Programming, General Structure of

Assembly Language

Writing Programs

Branch, Load and Store Instructions

Read-Only and Read/Write Memory

Multiple Register Load and Store

There are two items used in Assembly language Programming 1. Instructions 2. Directives (Assembler directives or Pseudo Instructions)

Instruction: In ARM Assembly language line will contains an instruction, directive or pseudo instruction.

Programmi Assembly ng

Dr. V.Vijayaraghavan, Assistant Professor-ECE,
VFSTR, Guntur

Programming Assembly

Dr. V.Vijayaraghavan, Assistant Professor-ECE,
VFSTR, Guntur

Assembly Programming.....

Dr. V.Vijayaraghavan, Assistant Professor-ECE,
VFSTR, Guntur

General Structure of Assembly Language

Dr. V.Vijayaraghavan, Assistant Professor-ECE,
VFSTR, Guntur

General Structure of Assembly Language.....

Dr. V.Vijayaraghavan, Assistant Professor-ECE,
VFSTR, Guntur

General Structure of Assembly Language.....

Dr. V.Vijayaraghavan, Assistant Professor-ECE,
VFSTR, Guntur

General Structure of Assembly Language.....

Dr. V.Vijayaraghavan, Assistant Professor-ECE,
VFSTR, Guntur

Writing Assembly Language Programs

Dr. V.Vijayaraghavan, Assistant Professor-ECE,
VFSTR, Guntur

Writing Assembly Language Programs.....

Dr. V.Vijayaraghavan, Assistant Professor-ECE,
VFSTR, Guntur

Branch Instructions

Dr. V.Vijayaraghavan, Assistant Professor-ECE,
VFSTR, Guntur

Conditional Execution:

Explanation with conditional execution is needed if asks Branch in 10marks question

Dr. V.Vijayaraghavan, Assistant Professor-ECE, VFSTR, Guntur

Branch Instructions.....

Dr. V.Vijayaraghavan, Assistant Professor-ECE,
VFSTR, Guntur

Branch Instructions.....

Dr. V.Vijayaraghavan, Assistant Professor-ECE,
VFSTR, Guntur

Branch Instructions.....

Dr. V.Vijayaraghavan, Assistant Professor-ECE,
VFSTR, Guntur

Branch Instructions.....

Dr. V.Vijayaraghavan, Assistant Professor-ECE,
VFSTR, Guntur

Branch Instructions.....

Dr. V.Vijayaraghavan, Assistant Professor-ECE, VFSTR, Guntur

Branch and Branch with Link and Exchange:

Branch and Branch with Link (B, BL)

Syntax: (or) Assembler Format: **B {L} {cond} <target address>**

➢ Branch (B) instruction is similar to Jump instruction (no need to return)

➢ Branch with Link (BL) instruction is similar to CALL instruction (need to return)

Binary encoding:

Ex: BNE loop1

➢ The range of the branch instruction is +/- 32 Mbytes.

Dr. V.Vijayaraghavan, Assistant Professor-ECE, VFSTR, Guntur

Branch and Branch with Link and Exchange...

Dr. V.Vijayaraghavan, Assistant Professor-ECE,

VFSTR, Guntur

Branch and Branch with Link and Exchange...

Branch, Branch with Link and exchange (BX, BLX)

Syntax: (or) Assembler Format:

1. B {L} X {cond} Rm

2. BLX <target address>

These instructions are available on ARM chips which support the Thumb (16-bit) instruction set, and are a mechanism for switching the processor to execute Thumb instructions or for returning symmetrically to ARM and Thumb calling routines. A similar Thumb instruction causes the processor to switch back to 32-bit ARM instructions. BLX is available only on ARM processors that support architecture v5T.

Dr. V.Vijayaraghavan, Assistant Professor-ECE, VFSTR, Guntur

Branch and Branch with Link and Exchange... Binary encoding:

Ex: BX R0

The branch target is specified in a register, Rm. Bit[0] of Rm is copied into the T bit in the CPSR and bits[31:1] are moved into the PC:

- If Rm[0] is 1, the processor switches to execute Thumb instructions and begins executing at the address in Rm aligned to a half-word boundary by clearing the bottom bit.
- If Rm[0] is 0, the processor continues executing ARM

instructions and begins executing at the address in Rm aligned to a word boundary by clearing Rm[I:0].

Dr. V.Vijayaraghavan, Assistant Professor-ECE, VFSTR, Guntur

Load/Store Instructions

Dr. V.Vijayaraghavan, Assistant Professor-ECE,
VFSTR, Guntur

Load/Store Instructions.....

Dr. V.Vijayaraghavan, Assistant Professor-ECE,
VFSTR, Guntur

Load/Store Instructions.....

Dr. V.Vijayaraghavan, Assistant Professor-ECE,
VFSTR, Guntur

Load/Store Instructions.....

Dr. V.Vijayaraghavan, Assistant Professor-ECE,
VFSTR, Guntur

Load/Store Instructions.....

Dr. V.Vijayaraghavan, Assistant Professor-ECE,
VFSTR, Guntur

Load/Store Instructions.....

Dr. V.Vijayaraghavan, Assistant Professor-ECE,
VFSTR, Guntur

Load/Store Instructions.....

Dr. V.Vijayaraghavan, Assistant Professor-ECE,
VFSTR, Guntur

Load/Store Instructions.....

Dr. V.Vijayaraghavan, Assistant Professor-ECE,
VFSTR, Guntur

Data Transfer Instructions...

Single Word & Unsigned Byte Data

Transfer Instructions (LDR, STR)

These instructions are the most flexible way to transfer single bytes or words of data between ARM's registers and memory. Transferring large blocks of data is usually better done using the multiple register transfer instructions, and recent ARM processors also support instructions for transferring half-words and signed bytes.

The pre-indexed form of the instruction:

LDR | STR {Cond} {B} Rd, [Rn, <offset>] {!}

The post-indexed form of the instruction:

LDR | STR {Cond} {B} {T} Rd, [Rn], <offset>

LDR is 'load register', STR is 'store register'; the optional 'B' selects an unsigned byte transfer, the default is word; <offset> may be +/- <l2-bit immediate> or +/- Rm and ! selects write-back (auto-indexing) in the pre-indexed form.

Dr. V.Vijayaraghavan, Assistant Professor-ECE, VFSTR, Guntur

Single Word and Unsigned Byte Data Transfer and Half Word and Signed Byte Data Transfer Instructions...

Binary Encoding:

Ex: STR r0, [r2]

Dr. V.Vijayaraghavan, Assistant Professor-ECE, VFSTR, Guntur

Single Word and Unsigned Byte Data Transfer and Half Word and Signed Byte Data Transfer Instructions... **Half-Word and Signed Byte Data Transfer Instructions**

These instructions are used to transfer signed bytes or half words of data between ARM's registers and memory. The addressing modes available with these instructions are a subset of those available with the unsigned byte and word forms.

The pre-indexed form of the instruction:

LDR | STR {Cond} H | SH | SB Rd, [Rn, <offset>] { ! }

The post-indexed form of the instruction:

LDR | STR {Cond} H | SH | SB Rd, [Rn], <offset>

These instructions are very similar to the word and unsigned byte transfer, but here the immediate offset is limited to eight bits and The S and H bits define

the type of the operand to be transferred. Since there is no difference between storing signed and unsigned data, the only relevant forms of this instruction format are:

• Load signed byte, signed half-word or unsigned half-word.

• Store half-word.

Dr. V.Vijayaraghavan, Assistant Professor-ECE, VFSTR, Guntur

Single Word and Unsigned Byte Data Transfer and Half Word and Signed Byte Data Transfer Instructions...

Binary Encoding:

Ex: LDRSH r0, [r1], #2

Dr. V.Vijayaraghavan, Assistant Professor-ECE, VFSTR, Guntur

and Read only Read/Write Memory

The two memory areas defined by the

compiler are

- ‘**Readonly**’for code, and ‘**Read/write**’for data.

▪ Usually this corresponds to ROM and RAM in a physical system.

▪ RAM is used for intermediate results, for temporary storage, etc., as this is volatile memory.

▪ We can store data permanently in the readonly memory, process it and copy it in RAM.

▪ In the readonly memory, data is written using directives like DCD, DCW, etc. From there, it is copied to readwrite memory using load and store instructions

Dr. V.Vijayaraghavan, Assistant Professor-ECE,
VFSTR, Guntur

Read only and Read/Write Memory.....

Dr. V.Vijayaraghavan, Assistant Professor-ECE, VFSTR, Guntur

In Example 10.15, three memory areas have been defined: one in readonly, and two in readwrite memory. What is accomplished is just the transfer of a word from readonly memory to readwrite memory.

In readwrite memory, one part is a space of 60 bytes. The next is a word space which is initialized to 0. After the execution of the program, the number **653451134** is copied to both these spaces.

The number is specified in decimal **653451134** Its equal Hexa value is **26F2DF7E**

Read only and Read/Write Memory.....

Dr. V.Vijayaraghavan, Assistant Professor-ECE,
VFSTR, Guntur

Read only and Read/Write Memory.....

Example 10.16 uses many of the programming aspects that we have been discussing so far. Let's have a look at the important features of this program.

▪ There is an ASCII string written in readonly memory using the DCB directive. Such a string is enclosed in double quotes and each character is a byte.

▪ One readonly and one read/write memory areas have been defined.

▪ After the ASCII string, a 0 is used as a terminating character. The arrival of this 0 in R2 is used to check whether the required transfer of the string is done.

▪ The instructions for loading and storing are suffixed by ‘B’ which indicates that only a byte is to be transferred.

▪ Post-indexed mode of addressing is used for load and store. The addresses need to be incremented only by 1, as only a byte is transferred.

▪ The instructions for loading and storing are in a procedure named COPY. The procedure is called by the BL instruction which does branching and also copies the

current PC to the link register. The last line of the procedure is copying the LR back to PC. This constitutes the ‘return’to the main program.

Dr. V.Vijayaraghavan, Assistant Professor-ECE, VFSTR, Guntur

Multiple Register Load and Store

An advanced (or let's say an extended) form of loading and storing, wherein multiple registers are involved. But only data in the form of words (32 bits) can be handled by these instruction. The mnemonic of multiple load and store is LDM/STM.

The LDM Instruction

LDM*{cond} address-mode* Rn{!},reg-list {^}

▪ Rn is the base register for the load operation. The address stored in this register is the starting address for the load operation.

▪ There can be a number of modes for specifying the address,

▪ ***register-list*** is a comma-delimited list of symbolic register names and register ranges enclosed in braces. There must be at least one register in the list. Register ranges are specified with a dash. For example, {R0-R5, R9} is a list.

▪ The ! option is for 'write back' and the ^ option is relevant for interrupts. Write back is not to be specified if the base register Rw is in ***register-list.***

Dr. V.Vijayaraghavan, Assistant Professor-ECE, VFSTR, Guntur

Multiple Register Load and Store.....

Dr. V.Vijayaraghavan, Assistant Professor-ECE, VFSTR, Guntur

Multiple register load means that multiple memory locations are to be accessed, and loaded into multiple registers. There is a 'base register' acting as a pointer for the first memory location to be

accessed. This register is then incremented or decremented to point to the next memory addresses.

Multiple Register Load and Store.....

There are four options for handling this. The base register can be incremented or decremented by 4 (one word needs four addresses) for each register in the operation, and the increment or decrement can occur before or after the operation. The suffixes for these options are as follows:

IA - increment after IB - increment before DA - decrement after DB - decrement before

Consider the instruction LDM DA R0, {R4-R9} The base register here is R0. Let us assume it holds the number 0x45000000. The operation of this instruction is that the 32-bit word at that address is pointed by R0, and that word copied to R4. Then the address is decremented to point to the next word. So the new address is [R0-4], and this word is copied to R5. The sequence of decrementing the address and loading data from memory is done for the registers R4, R5, R6, R7, R8 and R9. This single instruction replaces six LDR instructions. Is there

any advantage in this? As far as execution is concerned, it is 'No'. All the six load operations have to be done. But note that only one 'instruction fetch*cycle is needed for the six load operations together. So there is definitely some savings in terms of time.

Dr. V.Vijayaraghavan, Assistant Professor-ECE, VFSTR, Guntur

Multiple Register Load and Store..... There are four options for handling this.

The base register can be incremented or decremented by 4 (one word needs four addresses) for each register in the operation, and the increment or decrement can occur before or after the operation. The suffixes for these options are as follows:

IA - increment after , IB - increment before , DA - decrement after , DB - decrement before

Consider the instruction LDMDA R0, {R4-R9} The base register here is R0. Let us assume it holds the number 0x45000000. The operation of this instruction is that the 32-bit word at that address is pointed by R0, and that word copied to R4. Then the address is decremented to point to the next word. So the new address is [R0-4], and this word is copied to R5. The sequence of decrementing the address and loading data from memory is done for the registers R4, R5, R6, R7, R8 and R9. This

single instruction replaces six LDR instructions. Is there any advantage in this? As far as execution is concerned, it is 'No'. All the six load operations have to be done. But note that only one 'instruction fetch*cycle is needed for the six load operations together. So there is definitely some savings in terms of time.

What Is the difference in operation of the following instruction? LDMIA R10,{ R9, R1 - R5} Here the base address is in R10, and after each data transfer, it is incremented by 4. In the destination register list, R9 is specified first, but the processor has a particular way of handling the list. The lowest register will always be loaded from the lowest address in memory, and the highest register from the highest address. Here R1 gets the data in the address pointed by R10.

Dr. V.Vijayaraghavan, Assistant Professor-ECE, VFSTR, Guntur

Load/Store Instructions.....

Dr. V.Vijayaraghavan, Assistant Professor-ECE,
VFSTR, Guntur

Load/Store Instructions.....

Dr. V.Vijayaraghavan, Assistant Professor-ECE,
VFSTR, Guntur

Load/Store Instructions.....

Dr. V.Vijayaraghavan, Assistant Professor-ECE,
VFSTR, Guntur

Load/Store Instructions.....

Dr. V.Vijayaraghavan, Assistant Professor-ECE,
VFSTR, Guntur

Load/Store Instructions.....

Dr. V.Vijayaraghavan, Assistant Professor-ECE,
VFSTR, Guntur

Load/Store Instructions.....

Dr. V.Vijayaraghavan, Assistant Professor-ECE,
VFSTR, Guntur

Load/Store Instructions.....

Dr. V.Vijayaraghavan, Assistant Professor-ECE,
VFSTR, Guntur

Load/Store Instructions.....

Dr. V.Vijayaraghavan, Assistant Professor-ECE,
VFSTR, Guntur

Load/Store Instructions.....

Dr. V.Vijayaraghavan, Assistant Professor-ECE,
VFSTR, Guntur

Load/Store Instructions.....

Dr. V.Vijayaraghavan, Assistant Professor-ECE,
VFSTR, Guntur

Load/Store Instructions.....

Dr. V.Vijayaraghavan, Assistant Professor-ECE,
VFSTR, Guntur

Multiple Register Transfer Instructions:

The ARM multiple register transfer

instructions allow any subset (or all) of the 16 registers visible in the current operating mode to be loaded from or stored to memory.

The normal form of the instruction is:

LDM | STM {Cond} <add mode> Rn {!}, <registers list>

The register list in the bottom 16 bits of the instruction includes a bit for each visible register, with bit 0 controls r0, bit 1 controls r1, and so on up to bit 15 which controls the transfer of the PC.

The base address will be incremented (U = 1) or decremented (U = 0) before (P = 1) or after (P = 0) each word transfer.

Dr. V.Vijayaraghavan, Assistant Professor-ECE, VFSTR, Guntur

Multiple Register Transfer Instructions...

Binary Encoding:

Ex: STMFD r13 !, {r0-r2, r14}

Dr. V.Vijayaraghavan, Assistant Professor-ECE, VFSTR, Guntur

Binary Encoding:

Ex: SWPB r1, r1, [r0]

Dr. V.Vijayaraghavan, Assistant Professor-ECE, VFSTR, Guntur

Swap Instructions:

Swap Memory and Register Instructions (SWP)

Swap instructions combine a load and a store of a word or an unsigned byte in a single instruction. These instructions are little used outside their role in the construction of semaphores.

Syntax: SWP {Cond} {B} Rd, Rm, [Rn]

Software Interrupt: Software Interrupt (SWI)

The software interrupt instruction is used for calls to the operating system and is often called a 'supervisor call'. It puts the processor into supervisor mode and begins executing instructions from address 0x08.

Syntax: SWI {cond} <24 bit immediate>

The 24-bit immediate field does not influence the operation of the instruction but may be interpreted by the system code. 1. Save the address of the instruction after the SWI (PC) into r14_svc. 2. Save the CPSR into SPSR_svc. 3. Enter supervisor mode & disable IRQ by setting CPSR[4:0] to 100112 and CPSR[7] to I. 4. Set the PC to 08_{16} and begin executing the instructions there.

To return to the instruction after the SWI the system routine must not only copy r14_svc back into the PC, but it must also restore the CPSR from SPSR_svc.

Binary Encoding:

Dr. V.Vijayaraghavan, Assistant Professor-ECE, VFSTR, Guntur

The Thumb Instruction Set:

The Thumb instruction set addresses the issue of code density. It may be viewed as a compressed form of a subset of the ARM instruction set. Thumb instructions map onto ARM instructions, and the Thumb programmer's model maps onto the ARM programmer's model. Implementations of Thumb use dynamic decompression in an ARM instruction pipeline and then instructions execute as standard ARM instructions within the processor.

Thumb is not a complete architecture. Thumb is fully supported by ARM development tools, and an

application can mix ARM and Thumb subroutines flexibly to optimize performance or code density on a routine-by-routine basis.

The Thumb bit in the CPSR

ARM processors which support the Thumb instruction set can also execute the standard 32-bit ARM instruction set, and the interpretation of the instruction stream at any particular time is determined by bit 5 of the CPSR, the T bit. If T is set the processor interprets the instruction stream as 16-bit Thumb instructions, otherwise it interprets it as standard ARM instructions.

Dr. V.Vijayaraghavan, Assistant Professor-ECE, VFSTR, Guntur

The Thumb Instruction Set:

All Thumb instructions are 16-bits long –ARM instructions are 32-bits long

Most Thumb instructions are executed

unconditionally –All ARM instructions are executed conditionally

Many Thumb data processing instructions use a **2-address** format (the destination register is the same as one of the source registers) –ARM use 3-address format

Thumb instruction are less regular than ARM instruction formats, as a result of the dense encoding

Thumb properties –Thumb requires **70%** space of the ARM code –Thumb uses **40%** more instructions than the ARM code –With 32-bit memory, the ARM code is **40%** faster than the Thumb code –With 16-bit memory, the Thumb code is **45%** faster than the ARM code –Thumb uses **30%** less external memory power than ARM code

Dr. V.Vijayaraghavan, Assistant Professor-ECE, VFSTR, Guntur

The Thumb Instruction Set...

Thumb entry

ARM cores start up, after reset, executing ARM

instructions. The normal way they switch to execute Thumb instructions is by executing a Branch and Exchange instruction (BX).

Thumb exit An explicit switch back to an ARM instruction stream can be caused by executing a Thumb BX instruction. ARM instruction stream takes place whenever an exception is taken.

The Thumb programmer's model The instruction set gives full access to the eight 'Low' general purpose registers r0 to r7, and makes extensive use of r13 to r15 for special purposes:

- r13 is used as a stack pointer (SP).
- r14 is used as the link register (LR).
- r15 is the program counter (PC).

Dr. V.Vijayaraghavan, Assistant Professor-ECE, VFSTR, Guntur

The Thumb Instruction Set...

Thumb-ARM similarities

All Thumb instructions are 16 bits long.

• The load-store architecture with data processing, data transfer and control flow inst.

• Support for 8-bit byte, 16-bit half-word and 32-bit word data types.

• A 32-bit unsegmented memory.

Thumb-ARM Differences

• Most Thumb instructions are executed unconditionally.

• Many Thumb data processing instructions use a 2-address format.

• Thumb instruction formats are less regular than ARM instruction formats.

Thumb branch instructions

The ARM instructions have a large (24-bit) offset field which clearly will not fit in a 16-bit instruction format. Therefore the Thumb instruction set includes various ways of subsetting the functionality.

Dr. V.Vijayaraghavan, Assistant Professor-ECE, VFSTR, Guntur

Typical uses of branch instructions include: 1. short conditional branches to control

(for example) loop exit; 2. medium-range unconditional branches to

'goto' sections of code; 3. long-range subroutine calls.

B<cond> <label>; format1 - Thumb target B <label> ; format2 - Thumb target BL <label> ; format3 - Thumb target BLX <label> ; format3a - ARM target B{L}X Rm ; format4 - ARM or Thumb target

Dr. V.Vijayaraghavan, Assistant Professor-ECE, VFSTR, Guntur

The Thumb Instruction Set...**Thumb branch instruction binary encodings:**

The Thumb Instruction Set...

Thumb Software Interrupt Instruction

The Thumb software interrupt instruction behaves exactly like the ARM equivalent and the exception entry sequence causes the processor to switch to ARM execution.

Thumb software interrupt binary encoding:

This instruction causes the following actions:

• The address of the next Thumb instruction is saved in r14_svc.

• The CPSR is saved in SPSR_svc.

• The processor disables IRQ, clears the Thumb bit and enters supervisor mode by modifying the relevant bits in the CPSR.

• The PC is forced to address 0x08. The ARM instruction SWI handler is then entered. The normal return instruction restores the Thumb execution

state.

Syntax: SWI <8-bit immediate>

Dr. V.Vijayaraghavan, Assistant Professor-ECE, VFSTR, Guntur

The Thumb Instruction Set...

Thumb Data Processing Instructions:

Thumb data processing instructions comprise a highly optimized set of fairly complex formats covering the operations most commonly required by a compiler.

Dr. V.Vijayaraghavan, Assistant Professor-ECE, VFSTR, Guntur

The Thumb Instruction Set...

These instructions all map onto ARM data processing

(including multiply) instructions. Although ARM supports a generalized shift on one operand together with an ALU operation in a single instruction, the Thumb instruction set separates shift and ALU operations into separate instructions

Dr. V.Vijayaraghavan, Assistant Professor-ECE, VFSTR, Guntur

Thumb Implementation:

The Thumb instruction set can be incorporated into a 3-stage pipeline ARM processor macrocell with relatively minor changes to most of the processor logic. The biggest addition is the Thumb instruction decompressor in the instruction pipeline; this logic translates a Thumb instruction into its equivalent ARM instruction.

The Thumb decompressor performs a static translation from the 16-bit Thumb instruction into the equivalent 32-bit ARM instruction. This involves performing to translate the major and minor opcodes, zero-extending the 3-bit register to give 4-bit specifiers and mapping

other fields across as required.

- Since the only conditional Thumb instructions are branches, the condition 'always' is used in translating all other Thumb instructions.
- Thumb data processing instruction should modify the condition codes in the CPSR is implicit in the Thumb opcode; this must be made explicit in the ARM instruction.
- The Thumb 2-address format can always be mapped into the ARM 3-address format by replicating a register specifier. The simplicity of the decompression logic is crucial to the efficiency of the Thumb instruction set.

Dr. V.Vijayaraghavan, Assistant Professor-ECE, VFSTR, Guntur

Thumb Implementation...

The Thumb instruction decompressor organization

Dr. V.Vijayaraghavan, Assistant Professor-ECE, VFSTR, Guntur

Thumb Implementation...

Thumb to ARM instruction mapping

Dr. V.Vijayaraghavan, Assistant Professor-ECE, VFSTR, Guntur

Thumb Applications:

Thumb instructions are 16 bits long and

encode the functionality of an ARM instruction in half the number of bits, but since a Thumb instruction typically has less semantic content than an ARM instruction, a particular program will require more Thumb instructions than it would have needed ARM instructions. The ratio will vary from program to program, but in a typical example Thumb code may require 70% of the space of ARM code.

Thumb properties

- The Thumb code requires 70% of the space of the ARM code.
- The Thumb code uses 40% more instructions than the ARM code.
- With 32-bit memory, the ARM code is 40% faster than the Thumb code.
- With 16-bit memory, the Thumb code is 45%

faster than the ARM code.

• Thumb code uses 30% less external memory power than ARM code.

Dr. V.Vijayaraghavan, Assistant Professor-ECE, VFSTR, Guntur

Thumb Applications...

So where performance is all-important, a system should use 32-bit memory and run ARM code. Where cost and power consumption are more important, a 16-bit memory system and Thumb code may be a better choice.

Thumb Systems

• A high-end 32-bit ARM system may use Thumb code for certain non-critical routines to save power or memory requirements.

• A low-end 16-bit system may have a small amount of on-chip 32-bit RAM for critical routines running

ARM code, but use off-chip Thumb code for all non-critical routines.

➢ Thumb code will enable off-chip ROMs to give good performance on an 8- or

16-bit bus, saving cost and improving battery life.

➢ Mobile telephone and pager applications incorporate real-time digital signal

processing (DSP) functions that may require the full power of the ARM.

Dr. V.Vijayaraghavan, Assistant Professor-ECE, VFSTR, Guntur

Writing Simple Assembly Language Programs:

ARM Program:

'Hello World' program

Dr. V.Vijayaraghavan, Assistant Professor-ECE, VFSTR, Guntur

Writing Simple Assembly Language Programs...

'Hello World' program Thumb Program:

Dr. V.Vijayaraghavan, Assistant Professor-ECE, VFSTR, Guntur

Dr. V.Vijayaraghavan, Assistant Professor-ECE, VFSTR, Guntur

- **Lyla B. Das, “Embedded Systems: An Integrated Approach”, 1st Edition,**

Pearson, 2012. ❖ Raj Kamal, “Embedded Systems Architecture, Programming and Design”, 2nd Edition, McGraw Hill, 2009. ❖ Kenneth J. Ayala, “The 8051 microcontroller : Architecture, Programming”, 3rd Edition, Thomson Learning, 2007. ❖ David E. Simon, “An Embedded Software Primer”, 1st Edition, Pearson, 2008. ❖ Marilyn Wolf, “Computers as Components - Principles of Embedded Computing System Design”, 3rd Edition, Morgan Kaufmann Publisher (Elsevier), 2012. ❖ Jean J. Labrosse, “Embedded System Building Blocks: Complete and Ready-to-Use Modules in C”, 2nd Edition, CRC Press, 1999. ❖ Frank Vahid and Tony Givargis, “Embedded System Design: A Unified Hardware/Software Introduction”, 3rd Edition, John Wiley & Sons, 2006. ❖ K.V.K.K. Prasad, “Embedded Real-Time Systems: Concepts, Design & Programming”, Dream Tech Press, 2005. ❖ Steve Furber, “ARM System on Chip Architecture”, Addison Wesley, 2nd Ed., 2000. ❖ Andrew N Sloss, Dominic Symes and Chris Wright, “ARM system developers guide”, Morgan Kaufmann Publisher-Elsevier, 2008.

 2 October 2019 **Dr. V.Vijayaraghavan, Assistant Professor-ECE, VFSTR, Guntur**