ARM7 Note

The ARM7 is part of the Advanced RISC Machines (ARM) family of general-purpose 32-bit microprocessors, which offer very low power consumption and price for high-performance devices. The architecture is based on Reduced Instruction Set Computer (RISC) principles, and the instruction set and related decode mechanism are much simpler than microprogrammed Complex Instruction Set Computers. This results in a high instruction throughput and impressive real-time interrupt response from a small and cost-effective chip. • The instruction set comprises eleven basic instruction types: • Two of these make use of the on-chip arithmetic logic unit, barrel shifter, and multiplier to perform high-speed operations on the data in a bank of 31 registers, each 32 bits wide; • Three classes of instruction control data transfer between the memory and the registers, one optimized for flexibility of addressing, another for rapid context switching, and the third for swapping data; • Three instructions control the flow and privilege level of execution; and • Three types are dedicated to the control of external coprocessors, which allow the functionality of the instruction set to be extended offchip in an open and uniform way. • The ARM instruction set is a good target for compilers of many different high-level languages. Where required for critical code segments, assembly code programming is also straightforward, unlike some RISC processors, which depend on sophisticated compiler technology to manage complicated instruction interdependencies. • Pipelining is employed so that all parts of the processing and memory systems can operate continuously. Typically, while one instruction is being executed, its successor is being decoded, and a third instruction is being fetched from memory. The memory interface has been designed to allow the performance potential to be realized without incurring high costs in the memory system. Speed critical control signals are pipelined to allow system control functions to be implemented in standard low-power logic, and these control signals facilitate the exploitation of the fast local access modes offered by industry standard dynamic RAMs. • ARM7 has a 32-bit address bus. All ARM processors share the same instruction set, and ARM7 can be configured to use a 26-bit address bus for backwards compatibility with earlier processors. • ARM7 is a fully static CMOS implementation of the ARM, which allows the clock to be stopped in any part of the cycle with extremely low residual power consumption and no loss of state.

A programmer can think of an ARM core as functional units connected by data buses, as shown in Figure, where the arrows represent the flow of data, the lines represent the buses, and the boxes represent either an operation unit or a storage area. • The figure shows not only the flow of data but also the abstract components that make up an ARM core. • Data enters the processor core through the Data bus. The data may be an instruction to execute or a data item. The figure shows a Von Neumann implementation of the ARM—data items and instructions share the same bus. In contrast, Harvard implementations of the ARM use two different buses. • The instruction decoder translates instructions before they are executed. Each instruction executed belongs to a particular instruction set. • The ARM processor, like all RISC processors, uses a load-store architecture. This means it has two instruction types for transferring data in and out of the processor: load instructions copy data from memory to registers in the core; conversely, the store instructions copy data from registers to memory. • There are no data processing instructions that directly manipulate data in memory. Thus, data processing is carried out solely in registers. • Data items are placed in the register file—a storage bank made up of 32-bit registers. • Since the ARM core is a 32-bit processor, most instructions treat the registers as holding signed or unsigned 32-bit values. The sign extend hardware converts signed 8- bit and 16-bit numbers to 32-bit values as they are read from memory and placed in a register

Data items are placed in the register file—a storage bank made up of 32-bit ARM instructions typically have two source registers, Rn and Rm, and a single result or destination register, Rd. Source operands are read from the register file using the internal buses A and B, respectively. • The ALU (arithmetic logic unit) or MAC (multiply-accumulate unit) takes the register values Rn and Rm from the A and B buses and computes a result. • Data processing instructions write the result in Rd directly to the register file. Load and store instructions use the ALU to generate an address to be held in the address register and broadcast on the Address bus. • One important feature of the ARM is that register Rm alternatively can be preprocessed in the barrel shifter before it enters the ALU. Together the barrel shifter and ALU can calculate a wide range of expressions and addresses. • After passing through the functional units, the result in Rd is written back to the register file using the Result bus. For load and store instructions, the incrementer updates the address register before the core reads or writes the next register value from or to the next sequential memory location. The processor continues executing instructions until an exception or interrupt changes the normal execution flow

ARM7TDI Architecture

Features of ARM7TDMI • ARM7 has 32 bits of MCU & ALU. • ARM7 has 32 bits of Data bus with aligned memory space. {Means with each machine cycle, it will execute 32bits via a data bus. Here aligned addressing is done.} • ARM7 has all the instructions with 32 bits. • ARM7 has 32 bits of Address Bus. {So we can interface 4GB memory with it directly.} • ARM7 follows Von Neuman Model. {So 4GB memory will be common for code and data.} with the latest versions, they have switched to Harvard Architecture. • ARM7 has three stages of Pipelining: Fetch, Decode, Execute • ARM7 has 37 registers of 32 bits. At a time, 16 registers are available {R0 - R15). • ARM7 has LOAD–STORE Architecture. {Means for LOAD and STORE operations, we have to use separate instructions.} ARM7TDMI Core Diagram • ARMZ has seven operating Modes. • ARMZ has seven interrupts / Exceptions. • ARM7 has seven Addressing Modes. • ARM7 has 6 data formats [Signed/Unsigned] {8 bits-Byte, 16 bitsHalf word, 32 bits-Word} • ARM7TDMI: • T - Thumb Instructions • D - Hardware Debugging • M-Long Multiply instruction and • I - ICE Microcells for Breakpoints and watch-points in the program.

Operations : ARM7 performs mainly three kinds of operation • Execution of the instruction • Read data from the memory • Write data to the memory ARM7TDMI Core Diagram Address Data Bus - 32 Bits : By 32 bits, it can access a 4GB Memory address. As it is Von Neumann Model, Memory is common for Code & Data. PC - Program Counter - R15 - 32 Bits : PC indicates the address of the next instruction. PC is periodically incremented after every instruction is fetched. Since all instructions are of 32 bits, the lowest two bits are insignificant as all the instructions are at aligned locations (Multiple of 4). Data Bus - 32 Bits : This Bus is used for Fetch instruction opcode, Read Data & Write Data. Instruction Opcode will be given to the Instruction decoder for identification of operation. Data read on the data bus can be of 8bits, 16bits & 32bits. It will be given to Sign Extend before storing into any register of 32bits. ALU – 32 Bits : It is used to perform Arithmetic & Logical operations. It will take operands from registers only, and it will store results in the register only; along with that, it will update the flag register.

Barrel Shifter : It is used to pre-shift operand before it is given to ALU. It is a combinational logic shifter; it can shift multiple shifts in one clock cycle, Here we can pre-scale the operand. For example, if we shift left by 1 bit, then we multiply the operand by 2, and if we shift right by 1 bit, then we divide the operand by 2. It can be used with a single register also for Shifting and Rotating the data of the register. It can also be used for scaled addressing. Example of Base with scaled register indirect addressing mode. LDR R0, (R1, R2, LSL#4] After execution R0=Data in Memory [R1+R2 left shifted with 4 bit] Register File {R0-R15} : R0-R15 are 32bits GPRS. Among them, some are special registers {PC-R15, LR-R14, SP-R13}. ARM7 has a total of 37 Registers of 32 bits. Multiply Accumulate - MAC : It is used to perform Multiplication operations with accumulation. Example: MLA R0, R1, R2, R3 After execution R0 = R1*R2 + R3 Address Register : It is used to hold the operand address for load and store instructions. It has a dedicated incrementor for accessing data from subsequent locations. Here, the Role of the Incrementor is very essential in the Base index addressing mode {Indirect Addressing Mode}.

RISC Design Philosophy

RISC – Reduced Instruction Set Computer • RISC design initiated mainly for microcontrollers. • RISC works as per LOAD-STORE MODEL. {For LOAD and STORE operations, we need to execute separate instructions. LOAD & STORE can not be done in a single instruction. We can not perform logical & Arithmetic operations on memory data in a single instruction with a RISC processor.} • Most Instructions of RISC are registers based, so performance will be fast. • All instructions are of the same size, so parallel execution of instructions will be more efficient. • All instructions operate on the same data size, so parallel execution of instructions will be more efficient. • Fetch, Decode & Execute time for most instructions is of the same standards, so the pipeline will be more effective. • RISC has fewer addressing modes as most instructions are based on the register, so learning of the instruction set is simpler. • As most instructions are based on registers, RISC has more numbers of registers for programming. • Since most instructions are based on registers, it consumes less power to execute programs. • Most RISC architectures are scalable, like ARM, SPARK, etc. • RISC processor uses hardwire control, is smaller in size & less area on a chip. So many features are provided on-chip like Timer, ADC, DAC, IO Ports etc.

RISC Vs CISC

Parameters RISC CISC Full Form Reduced Instruction Set Computer Complex Instruction Set Computer Instruction Size Fixed Size Variable Size Instruction Fetch Time Same for all instruction Vary with respect to instructions Instruction Set Small & Simple Large and Complex Addressing Modes Less Modes as most instructions are based on registers More Modes as Complex instructions are available with different verities Numbers of Registers Many Few Complier Design Simple Complex Program Size Long {Weak code density} Small (Better Code density} Numbers of Operand Fixed {Mainly in Registers} Variable {Can be in Registers & Memory} Control Unit Hardwire controlled Micro Program Controlled Execution Speed Faster Slower Pipelining More Effective Less Effective {It has more bubbles due more memory based instructions} Processor More suitable for dedicated operations More suitable for verities of operations

MEMORY MAPPED I/O

Parameters Von Neumann Harvard Memory Data and Program {Code} are stored in same memory Data and Program {Code} are stored in different memory Memory Type RAM for Data & Code RAM for Data and ROM for Code Buses Common bus for Address & Data/Code Separate Buses for Address & Data/Code Program Execution Code is executed serially and takes more cycles Code is executed in parallel with data so it takes less cycles Data/Code Transfer Data or Code in one cycle Data and Code in One cycle Control Signals Less (Single memory R/W operation) More (Multiple memories R/W operations) Space It needs less Space It needs more space Cost Less Costly

Data Store Order

There are two ways of ordering the data while storing it into the Memory; Big Endian & Little Endian • Big Endian : Most Significant Byte is stored 1st in the memory, and the Last significant Byte is stored at the last in the memory. • Little Endian : Last Significant Byte is stored 1st in the memory, and the Most significant Byte is stored at last in the memory. • E.g., Suppose we want to store 12345678H data starting at address 1000H.There are two ways of ordering the data while storing it into the Memory; Big Endian & Little Endian • Big Endian : Most Significant Byte is stored 1st in the memory, and the Last significant Byte is stored at the last in the memory. • Little Endian : Last Significant Byte is stored 1st in the memory, and the Most significant Byte is stored at last in the memory. • E.g., Suppose we want to store 12345678H data starting at address 1000H. Add Data … … 1000H 12H 1001H 34H 1002H 56H 1003H 78H 1004H … … … Big Endian Add Data … … 1000H 78H 1001H 56H 1002H 34H 1003H 12H 1004H … … … Little Endian • As we also operate with stack memory in the CPU, we can remember this as per: • In Big Endian, Lower Address holds a higher Byte of data, and a Higher Address holds a lower Byte of data. • In Little Endian, Lower Address holds a Lower Byte, of data, and Higher Address holds Higher Byte of data. • Big Endian examples: Motorola 68xx, IBM Mainframe, TCP/IP etc. • Little Endian examples: Intel x86, AMD, PIC, etc

Advantages of Little Endian: • Easier for Multiplication & Addition of multiprecision numbers.

Data Types

Extended After Loading : After loading Data from the memory location, Data may have the size of 8-bits, 16-bits, or 32- bits. But that data must be extended into 32 bits, as it must be copied into a register and the size of the register is 32 bits.Unsigned numbers are extended by inserting 0's ahead number. Positive numbers are extended by inserting 0's ahead number and negative numbers are extended by inserting 1's ahead numbers. • Example : If Data = +7 = 0000 0111b then it will be extended as 0000 0000 0000 0000 0000 0000 0111b • Example : If Data = - 7 = 1111 1001b then it will be extended as 1111 1111 1111 1111 1111 1111 1111 1001b • Here, in binary for signed number, MSB shows the polarity of data. {+Ve or -Ve Data} In most of the computers negative number is accounted as per 2's complement of the given number in signed number representation.

Operating Modes

User Mode : This is the normal mode in which all the user programs are executed. It is the only non privileged mode. It has limited access to Memory, IO, and Flags. All other modes are entered through an interrupt. Fast Interrupt Mode : This mode is enabled when high priority interrupts come on nFIQ Pin. These interrupts should be served with minimum delay. By default, Nested interrupts are enabled in this Mode. Interrupt Mode : This mode is enabled when a normal interrupt comes on the nIRQ Pin. These interrupts will be served with some delay. Nested interrupts are allowed in this Mode. Supervisor Mode : This mode is enabled when we reset the system. It is used to execute the BIOS program. This mode can be invoked by the programmer with SWI {Software interrupt}. Abort Mode : This mode is entered when an unsuccessful attempt is made to access memory. Due to the protection mechanism, some memory locations are not accessible. Undefined Mode : This mode is entered when undefined instructions are attempted. This generally happens when coprocessor instruction is encountered, but the coprocessor is not available.

Programming Models and Registers

All the registers are orthogonal in ARM. Orthogonal means any instruction applied to R0 is also applicable to any register. • User/System Mode has R0-R15 and CPSR register. In total 17 registers. Main Program is there in user mode only. • Other Five modes have R0-R15, CPSR & SPSR registers; other modes are executed by some shorts of interrupt. So, other modes are served as ISR. • PC - Program Counter: It holds the address of the next instruction. • LR - Link Register: It holds the data of the PC during other modes of execution & Branch execution {CALL}. • CPSR - Current Program Status Register: It holds the status and control of the program. • SPSR - Saved Program Status Register: It holds the CPSR of User mode during other modes execution. • SP - Stack Pointer: It holds the address of the Top of stack memory. SP is separately provided to all the modes. • FIQ Register - R8 to R12: These are dedicated registers to provide fast service to interrupt. • So in total ARM7 have 17 User/System, 5 SP, 5 LR, 5 SPSR & 5 FIQ = 37 registers of 32 bits.

ARM7 Pieplining

ARM Pipelining • ARM7 uses 3 stage pipeline: Fetch, Decode & Execute. • ARM9 uses 5 stage pipeline: Fetch, Decode, Execute, Memory write & Register write. • ARM10 uses 6 stage pipeline: Fetch, Issue, Decode, Execute, Memory write & Register write. • Fetch, Decode & Execute has equal size of Machine Cycle. • Fetch: All the instruction of ARM7 has size of 32 bits. So, it will take only one machine cycle to fetch it as ARM7 has 32 bits data bus with aligned locations. • Decode: After fetch hardwire decode circuit will decode instructions in one machine cycle only. • Execute: Execution is done by ALU & MAC with respect to registers only. So execution will take one machine cycle only. • ARM7 has LOAD and STORE architecture. So, for loading the data from memory and storing data on to memory, there are separate instructions. • As ARM7 support 3 stage pipeline, PC always points 2 instructions ahead of the one being executed now. • Advantage: Fast Program Execution. • Disadvantage: Pipeline Fails in Branch Instruction execution.

Model of a Synchronous k-stages pipeline

Extending the Concept to Processor Pipeline • The same concept can be extended to hardware pipelines. • Suppose we want to attain k times speedup for some computation. • Alternative 1: Replicate the hardware k times -> cost also goes up k times. • Alternative 2: Split the computation into k stages -> very nominal cost increase. • Need for Buffering • In the hardware pipeline, need a Latch between successive stages to hold the intermediate results temporarily • The latches are made with master-slave flip-flops, and serve the purpose of isolating inputs from outputs. • The pipeline stages are typically combinational circuits. • When the Clock is applied, all latches transfer data to the next stage simultaneously.

Memory with ARM7

Memory Allocation in ARM • The ARM has 4GB of Memory space. • This memory space has address from 0000 0000H to FFFF FFFFH. • This 4GB memory space can be divided into five sections: 1. On Chip Peripherals and IO Registers: • This area is dedicated to General purpose IO - GPIO and Special Function Registers - SFR of peripherals such as Timers, Serial Communication, ADC etc. • Memory Mapped IO is used here. • It can vary chip to chip of ARM7. 2. On Chip Data SRAM: • It ranges from few KB to serval hundreds of KB. • It is used by data variables and STACK. • It can vary chip to chip of ARM7. • It's locations are well managed by C complier On Chip EEPROM: • It ranges from 1KB to serval hundreds of KB. • It is used store some program codes, Most often used for critical program. • It can vary chip to chip, it may not be available. 4. On Chip Flash ROM: • It ranges from few KB to several hundreds of KB. • It is used store program Space. • It can also be used to store some fixed data like ASCII & Look up table. • It is there under control of PC of ARM. • It can vary chip to chip. 5. OFF Chip DRAM: • It ranges from few MB to serval hundreds of MB. • It can be implemented by external interfacing with ARM. • User can interface OFF Chip DRAM based on the need of application

AMBA

Basics of AMBA • AMBA (Advanced Microcontroller Bus Architecture) is SoC Bus architecture for Microcontroller. • Bus Architecture Includes System Bus, {Address Bus + Data Bus + Control Bus} • AMBA provides the interface to functional blocks of the microcontroller. • AMBA is used to Minimize the silicon infrastructure, to support Modular system design & to make system technology independent. Possible Functional blocks on Board • One or More Microcontroller/Microprocessor. • Memory {SRAM, DRAM, EPROM, Flash Memory} • DSP {Digital Signal Processor} • DMA {Direct Memory Access} • USBS, SPI, 12C, IO ports, ADC/DAC, Timer. AMBA Standards • AHB - Advanced High Performance Bus • ASB - Advanced System Bus • APB - Advanced Peripheral Bus • ATB - Advanced Trace Bus • AXI-AMBA Extensible Interface APB • Advanced Peripheral Bus used for low BW Peripherals. • It has simple protocols and No pipeline support. • Used for read/write data from the bridge to the peripherals. • Bridge is Master & All the peripherals are slaves. • APB shares same signals for read & Write. • Burst transfer is not possible.AHB • Advanced High performance Bus uses for high Bandwidth interface. • Components can be DMA, DSP, System Memory, CPU etc. • It Supports Master Slave Concept. • Multi Master & Multi Slaves are supported. • Burst Transfer is possible.

Paging

Basics of Paging • It is used to convert the virtual address into a Physical Address. • By only physical addressing, with 32bits of addressing we can only interface 4GB only. • By virtual memory addressing, we can address more than physical space. • Example: ARMv8 has 52bits of virtual memory addressing, so with ARMv8, we can interface at max 4PB.

TLB && MMU

Translation Look aside Buffer - TLB • The Translation Lookaside Buffer (TLB) is a cache of recently accessed page translations in the MMU. • For each memory access performed by the processor, the MMU checks whether the translation is cached in the TLB. If the requested address translation causes a hit within the TLB, the translation of the address is immediately available. • Each TLB entry typically contains not just physical and Virtual Addresses but also attributes such as memory type, cache policies, access permissions, the Address Space ID (ASID), and the Virtual Machine ID (VMID).Translation Look aside Buffer - TLB • If the TLB does not contain a valid translation for the Virtual Address issued by the processor, known as a TLB miss, an external translation table walk or lookup is performed. • Dedicated hardware within the MMU enables it to read the translation tables in memory. • The newly loaded translation can then be cached in the TLB for possible reuse if the translation table walk does not result in a page fault. The exact structure of the TLB differs between implementations of the ARM processors. • If the OS modifies translation entries that may have been cached in the TLB, it is then the responsibility of the OS to invalidate these stale TLB entries. MMU converts virtual address into physical address with following advantages: • Protection Mechanism {provides protection at different privileged levels} • Mitigates Memory fragmentation {makes memory access linear} • Enhance the physical memory {Total possible memory is no longer depends on physical addressing} • Memory usage is scalable. {No fixed size required for any segments.} • Make access faster with the use of cache memory. • User don't need to separate code and data. {Code and data accessed by MMU} • MMU separates tasks with respect usage of memory

Cache Memory

What is Cache Memory? • It is SRAM. • Cache Memory holds recently & frequently accessed files. • Cache Memory may be there on chip or off chip. Basics of Cache Memory • Cache memory is small in size and it has fast data access. • The part of data and program are transferred from Primary or secondary memory to cache memory by OS. • If data from Primary Memory is mapped with the address inside cache memory then it is referred as 'Cache Hit' and If data from Primary Memory is not mapped with address inside cache then it referred as 'Cache Miss'. • If data is not available in cache memory then by different memory mapping scheme, data will be mapped to cache memory. Flush of Cache Memory • When CPU is accessing data from cache and if it is available with cache memory then operations will be super fast. • But if Outdated {No longer in use of core} data is there in cache then it has to get flush and there must be currently accessed data on cache memory for faster execution of programs. • Operating system OS updates currently accessed files into cache memory for faster execution of programs. • In order to execute program fast, OS will remove outdated data from cache is referred Flush of cache memory. Clear Cache Memory • If data is overloaded with cache memory, then program execution will be slower. • Important programs may not work as per the expectations of programmer. • By clearing cache memory, we can make execution fast. • Clear cache decision is taken by programmer not by OS. Advantages of Cache Memory • Less Access time. • Program execution is fast with Cache. • It stores the data for temporary usage. Disadvantages of Cache Memory • Limited size. • Costly

Explanation of the ARM7 Data Flow Model

The ARM7 architecture data flow model explains how instructions and data move within the processor during execution. Here's a breakdown of its major components and workings: 1. Instruction Fetch Unit Fetches instructions from memory.

Indian Institute of Information Technology Allahabad

Prayagraj, Uttar Pradesh – 211012, India

C2 Assessment (Odd Semester 2023-24)
Embedded System Design (ESD)

Instructions:

- Make the appropriate assumption(s) with an explanation.
- No need to explain the diagram in the form of a theory. No marks for it.
- It is recommended that explain the answer in the form of FlowChart or PseudoCode or BlockDiagram etc.

Sr.No.	Question	Marks
1	**Background:** In the current time scenario, autonomous electrical vehicles are available in the market. A stepper DC motor can be used for the wheel rotation of the vehicle. This motor can vary the rpm based on the duty cycle which is based on Pulse Width Modulation. When ON time width is more compared to OFF time width then motor rpm will increase as per the ratio and vice versa. **Problem:** You need to design an embedded system for the traffic signal which is based on an ARM microcontroller. This system will vary the duty cycle (ON and OFF pulse width) to control the rpm speed of the vehicle (motor rpm) as per the traffic rule. (a). Design the pipeline stages for the given problem only for ARM-7 architecture. (b). Represent the ARM Mode Switching through a diagram for the given problem only and based on your designed pipeline in the previous statement. (c). Define at least 5 'ARM Instructions' with 'Operation Definition' and its Meaning' that should be applied to the given problem (and as per the previous 2 statements). **Hint:** You can consider that traffic symbol information can be captured by the camera and required information can be extracted using image processing techniques. You can use a motor driver to control the duty cycle and accordingly, the pulse width can be varied by the microcontroller for the motor driver.	5+5+5