ARM Cortex M3: Overview & Programmer's Model

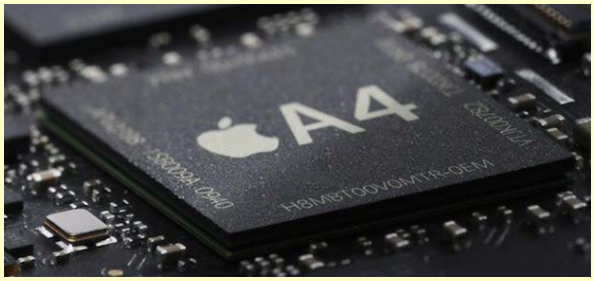
ECE 331, Spring 2013

Overview of Computing Systems

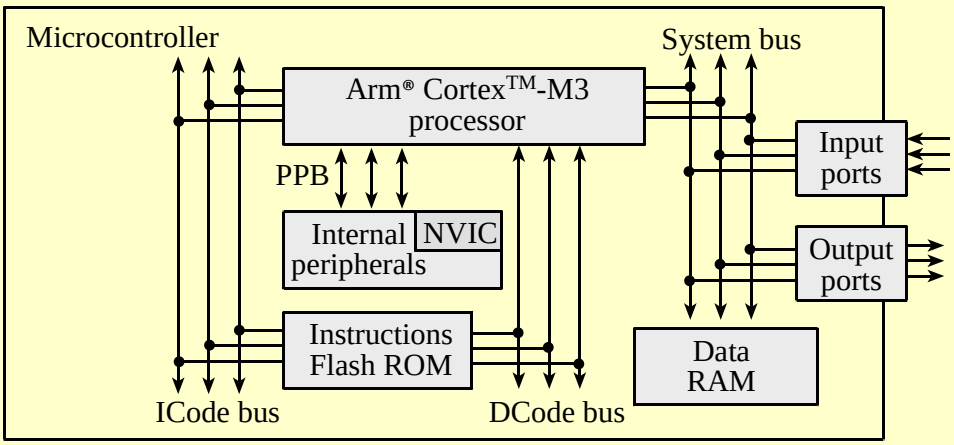
- **RISC**
  - ARM stands for **A**dvanced **R**ISC **M**achine
    - RISC = **R**educed **I**nstruction **S**et **C**omputer
  - Earliest work began in the late 60's/early 70's
- **RISC/ARM goals**
  - All instructions executed in single cycle
  - All instructions were the same size and had fixed format (32 bits in this case)
  - Simple to decode instructions
  - Easier to validate
  - Load/Store architecture. Data in external memory accessed with explicit instructions. All other operations (adds, subtracts, logic, etc) use only registers on the processor.
- **Feature creep**
  - Although it is a "reduced" instruction set, many instructions have been keeping up with the need for more demanding algorithms.
  - For instance, 31 instructions in Berkeley RISC-1, 46 in the second ARM processor, up to several hundred now.

Widespread Use Of ARM Today

- More than 10 billion ARM processors shipped so far.
  - Digital Cameras
  - Anti-lock disc brakes
  - Gameboy
  - Apple devices
  - Cell phones (Android)
  - So many more



Cortex-M3 Architecture



- Harvard Architecture: Separate data and instruction buses
- Cortex-M3 instruction set combines high performance typical of 32 bit processor with code density of 8 and 16 bit controllers
- Each 8-bit byte has unique address meaning the processor can read or write 8, 16, or 32 bit data

Features

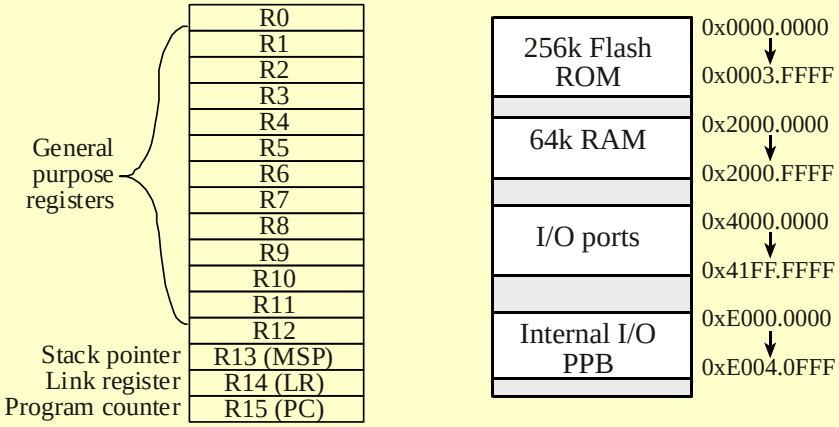
- Some features visible in the previous slide
- 32 bit core
  - 32-bit address space
  - 32-bit registers
  - 32-bit shifter and ALU
  - 32-bit memory transfer

Programmer's Model

- Registers: The most basic storage area on the chip. Can be used for data, timer, counter, addresses, etc.
  - 30 general-purpose registers (for loads and stores)
  - 6 status registers
  - A program counter
  - 37 total registers
- At one time…
  - 15 general purpose registers (r0-r14)
  - One or two status registers
  - Program counter (r15 or PC)
- All registers are 32 bits wide
- One thing many fail to understand is that these registers themselves occupy memory on the device
  - For instance, registers on ARM7TDMI are between 0x10000000-0x10000FFF

Registers

- What is a register?
  - High speed storage inside the processor
- R0-R12 are general purpose registers, contain either data or addresses
- R13 is stack pointer, points to top element of stack
- R14 is link register, used to store return location for functions (subroutines)
- R15 is the PC, points to the next instruction being fetched from memory



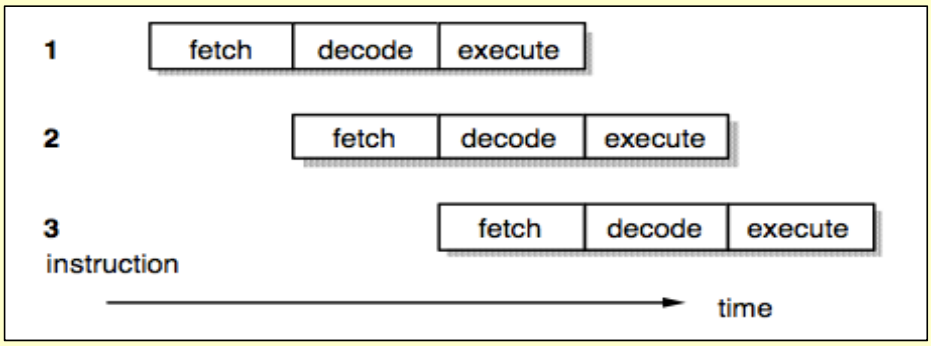
Special Purpose Registers

- R13, the Stack Pointer:
  - Holds the address of the stack in memory
  - Unique stack pointer in all modes (except system- shares with user)
- R14, the Link Register:
  - Subroutine return address link register
  - Unique link register in all modes (except system-shares with user)
- R15, Program counter (PC)
  - PC holds address of instruction being fetched.
  - Usually only used for long memory jumps or exception recovery
- Current Program Status Register (CPSR)
  - “State of the machine”
  - Allows programs to recover from exceptions or branch on results of an operation

Instruction Execution

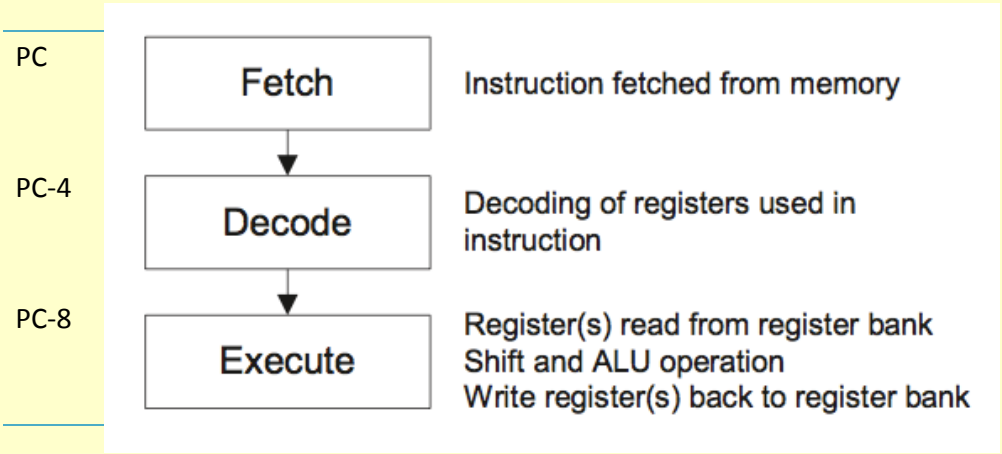
- Single thread, pipelined architecture
- At any given time (clock cycle)
  - one instruction being fetched
    - another being decoded
      - another being executed

Pipelining Instructions



- Fetch: Instruction fetched from memory
- Decode: Decoding of registers used in instruction
- Execute: Registers read from Register Bank, shift and ALU operation occur, write registers back to register bank
- PC always contains the address of the instruction currently being fetched (2 instructions past what is currently being executed). During pipelining, the pc increments twice before instruction one executes

Programmer's Model-Pipeline



Pipelining Example

- code

```
    LDR R1, =0x01
    ADD R1,R1,#4
    SUBEQ   R1,R1,#4
    SUB R1,R1,#7
    ADDMI   R1,R1,#10
    ADDS    R1,R1,#2
    ADDEQ   R1,R1,#3
DN  B   DN
```

- pipeline

Fetch	Decode	Execute
LDR R1, =0x01	-	-
ADD R1,R1,#4	LDR R1,=0x01	-
SUBEQ R1,R1,#4	ADD R1,R1,#4	LDR R1,=0x01
SUB R1,R1,#7	SUBEQ R1,R1,#4	ADD R1,R1,#4
ADDMI R1,R1,#10	SUB R1,R1,#7	SUBEQ R1,R1,#4
ADDS R1,R1,#2	ADDMI R1,R1,#10	SUB R1,R1,#7
ADDEQ R1,R1,#3	ADDS R1,R1,#2	ADDMI R1,R1,#10
B DN	ADDEQ R1, R1,#3	ADDS R1,R1,#2
-	B DN	ADDEQ R1, R1,#3
-	-	B DN

Number Representation

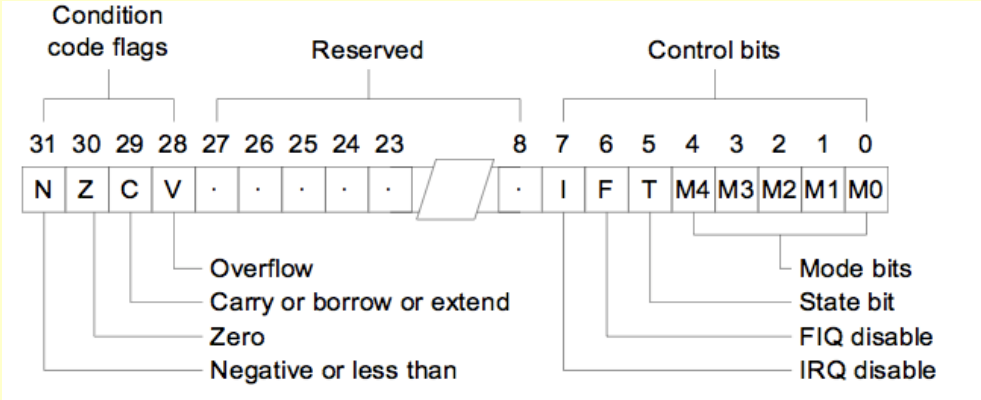
- Hex represented with prefix ‘0x’
  - Ex. 0x0FC1AB22
- Binary represented with suffix ‘y’
  - Ex. 11010010y
- Decimal requires no prefix or suffix.

Closed Brackets

- Closed brackets, [ ], treat what they contain as an address.
- Ex. Access data stored at memory

```
LDR  r1, =dataLoc  ;load the 32 bit address in r1
LDR  r2, [r1] ; r1 contains an address, this loads r2 with what is stored at the address
```

Program Status Register Format

- 

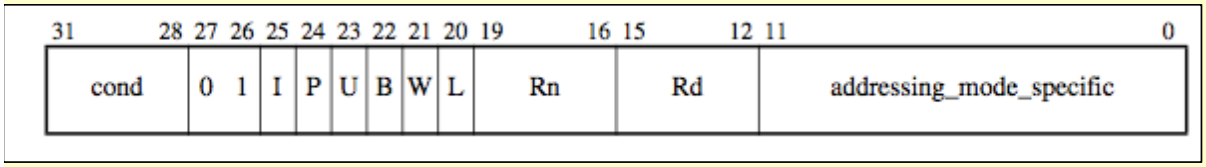
- I/F bit:
  - Active low interrupt set
- T bit:
  - Set to 1 if THUMB mode (uses 16 bit instructions)

Conditional Execution

- **Most instruction sets only allow branches to be taken conditionally.**
  - ALL ARM instructions have a condition field that determines whether or not the instruction would be executed.
  - Instructions that are not executed take up only 1 cycle - could be an advantage for long code.
  - Frequently less overhead than the usual branch or subroutine jump.
  - Ex. ADDEQ r0, r2,r5 ; if Z flag = 1, r0 = r2+r5

Load and Store Instruction Format

- 31st bit represents N flag, 30th bit represents Z flag, 29th bit represents C flag, 28th bit is the V flag
- I,P,U,W bits distinguish between different types of addressing modes. Wont get too specific but indicates whether immediate or non-immediate use of an operand
- L: distinguishes between a load (L =1), or store (L = 0)
- B: distinguishes between an unsigned byte (B=1) and a word (B=0)
- Rn: specifies the base register used by addressing_mode (R0-R15)
- Rd: specifies the register whose contents are to be loaded/stored



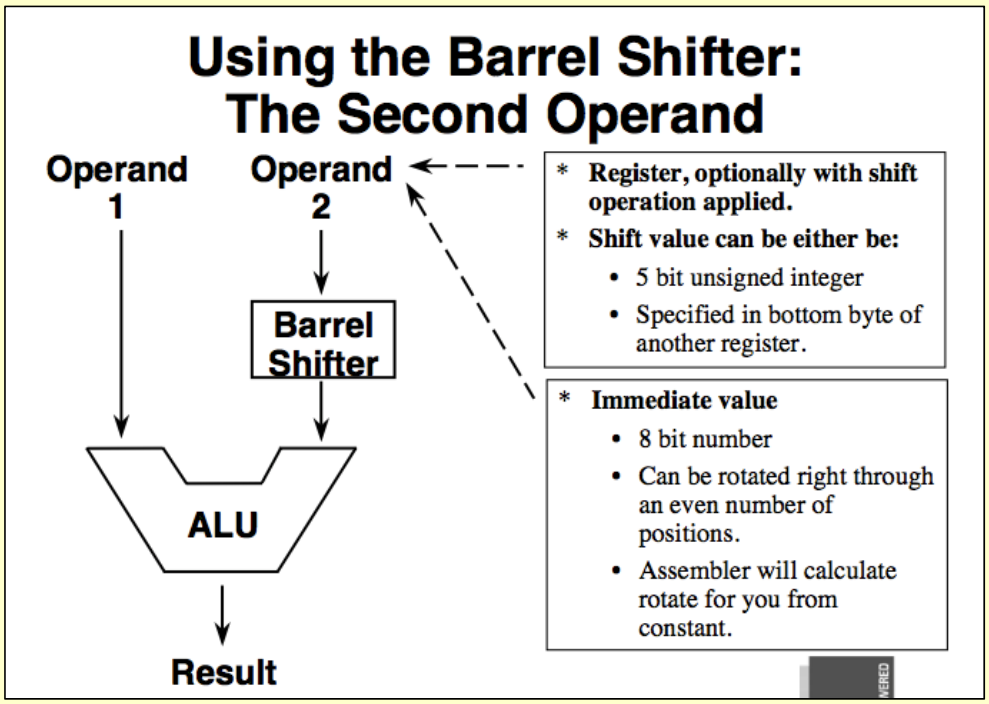
Operands

- Operands are the “variables” passed to the instruction.
- In many instructions, such as data processing instructions, instructions perform a specific operation on one or two operands.
- Operand 1 is always a register.
- Operand 2 is sent to the ALU by barrel shifter

Second Operand

- Shifted register
  - Amount to shift is contained in 5 bit instruction field
    - No overhead, shift is done free in one cycle
  - Shift is stored in bottom byte of a non-register PC
    - Takes extra cycle because ARM only has 2 read ports

Barrel Shifter



Big Vs. Little Endian

- Big Endian: Stores most significant byte of data at lower memory address
- Little Endian: Stores most significant byte of data at higher memory address
- Most ARM processors are biendian, can be configured to handle both
- Ex: Store 16 bit number 0x03E8 at locations 0x2000.0850 and 0x2000.0851

Big Endian	
Address	Data
0x2000.0850	0x03
0x2000.0851	0xE8

Little Endian	
Address	Data
0x2000.0850	0xE8
0x2000.0851	0x03