CH1 and Review

You are required to study the text for full definitions and explanations.

- Many important items from the forward were covered

- A microprocessor is interfaced to a bus or busses corresponding to control signals, address signals, and data signals, collectively called the system bus.
- firmware is the program the microprocessor runs. Typically, it is stored in a ROM. The instruction cycle includes fetching, decoding, and execution of instructions stored in the ROM.
- Upon boot-up, an instruction is fetched from a predetermined address. After every instruction, the instruction address must be updated by increment or jump. The instruction address is stored in a register called a program counter (PC).
- The set of instructions the microprocessor can decode and execute is called the instruction set.
- An embedded system program is typically designed to run continuously rather than execute and terminate with some result
- Watchdog Timer
    - Since crashing without automated recovery is typically not tolerated, a watchdog timer is a regular component in an embedded system. A watchdog timer's role is to detect system hang-up and issue a system reset signal after a predetermined amount of time. This automatic reset is avoided under normal operation by having the system software periodically reset the watchdog timer. A failure of the software to reset the watchdog timer in the allowed time span implied that a hang-up has occurred.
- Real-Time operation is a typical constraint on an embedded system /software. Concept based on meeting predetermined timing constraints rather than just executing as fast as possible.
- Soft Real-Time: not meeting timing constraints represents ***degraded performance***
- Hard Real-Time: not meeting timing constraints represents ***failure***
- Firm Real-Time: has a mixture of soft and hard constraints for various tasks
- Many real-time systems require definition, treatment/management, and management of formal tasks and processes (will formalize these later in the course)
- Need to develop methods for task cooperation to work together in a coordinated fashion and task communication to share data
- Polling or event-driven schemes can define the management and execution of processes and define how to interface with the external world
- Will learn the role of operating systems to facilitate these

Computer System Vocabulary:

- input
- output
- memory
- datapath
- control
- software
- firmware
- CPU
- bus, busses
- address signals
- data signals
- control signals

- bits
- signal width
- bus width
- address bus
- data bus
- control bus

Microprocessor Vocabulary:

- registers
- internal registers
- firmware
- datastore
- ROM
- RAM
- word size
- Microcomputer: complete computer systems that uses a microprocessor, though not typically referring to a mainframe
- Microcontroller: integrates, memory, IO, communications, and other peripherals and peripheral interfaces
- DSP: digital signal processor, microprocessor with special hardware and optimized for signal processing, typically lower cost and power for signal processing applications as compared to general
  purpose processor. Typically Harvard rather than von Neumann, so separate memory and busses for
  data and program

Flip-flop vs register vs latch

- Flop-flops are typically edge-triggered, updates synchronized to a sync. signal called a clock. The timing of input capture and output updates is determined by the occurrence of clock edge
- Latches outputs can respond immediately to data input changes, though involve an enable signal that is typically level-sensitive
- "register" typically refers to a multi-bit flip-flop

Numbers

- A collection of bits has no inherent meaning. The meaning comes from the interpretation and that is defined by the data-type and its storage specification

Binary

- Binary Number Representation
  - Signed Binary Number Representations
    -

One's complement (negation: INVERTED BITS,	MSBit is sign bit)
Two's complement (negation: INVERTED BITS + 1 ,	MSBit is sign bit)

Endianness

- "endianness" describes how to interpret and manipulate data, it is the order of bits or bytes in a word
-

	MSB down to	LSBit	
Big Endian	31 ...	0	Most significant bit first, or leftmost; least significant bit last, or rightmost

	LSB up to	MSBit	
Little Endian	0 ...	31	Least significant bit first, or leftmost; most significant bit last, or rightmost

- May specify order stored in registers, order in memory, order of bits or bytes sent over a communication channel like a serial port
- again, pay attention to word order versus bit order, can be **least** significant **bit** first and **most** significant **byte** first

Number of Bits of Resolution (updated)

- *Number of bits of resolution* relates to the amount of precision and range
  - is the number of levels that can be represented
  - can be given as the # of digits (bits if binary)
    - 3 bits of resolution means 2^3 unique levels can be represented
  - Real numbers must be rounded/truncated or otherwise mapped to one of these discrete and countable number of levels. The mapping is called quantization.
- Error Propagation (possibly more on this later)
  - Addition: error of operands adds
  - Multiplication: error of operands multiply

Instructions

- Address (noun): value used to indicate position in an array
- Instructions: used to indicate actions
- Operands provide parameters for action
- Arity refers to the number of operands an instruction or operation requires
  - e.g. y+z ; addition here is binary operator
  - Example references to arity: Unary operator, two-operand instruction, two-address instruction
  - Example:
    - x=y+z is a three-operand or three-address instruction
      - Has two sources or source addresses
      - Has one destination or destination address
- Typically instructions are encoded using a concatenation of fields (groups of bits with associated meaning)
- Common to have operation or op-code fields concatenated with source and/or destination fields

	Bit 31 ……...bit 0	
	operation or opcode | operand operation | op0 | op1 operation op2 | op1 |op1	

- Operands may refer to registers or other memory (or other addresses if using memory-mapped I/O)
- In most systems, if an operand is memory location the contents must be moved to registers to perform the computations
- Opcode design
  - RISC (Reduced Instruction Set Computers) require fewer bits for the op-code (leaves more bits for operands)
  - CISC (Complex Instruction Set Computers) require more bits for the op-code (leaves less bits for operands)

- Instruction Set Architecture (ISA) is the interface provided/presented to the programmer/compiler. It is everything that documents the use of the processor but not necessarily the internal details that are of no consequence to the programmer/compiler. Includes specification of op-codes, registers, operand formats, memory access methods, etc...
- Machine Language: collections of 0's and 1's that tell the computer what to do, op-codes and operands
- Assembly Language: programmers create code with instructions chosen from the supported instruction set
  Compiler converts this to machine code -- not quite 1-to-1 mapping to machine code but there is a very strong connection from architecture through machine language specification to assembly language and the ISA.

Ex:	add reg1 reg1 add reg1 memaddress1 add reg1 const1

  We can see "add" requires two operands, in this case we expect reg1 is a source and a destination. Which forms are supported in the assembly language usually relate to what hardware features are available and what machine code operations are supported. (some might support only reg operands, so the last operation would require a load of const1 to a register first)
- Compiled Code: high-level code is converted to assembly through compilation and then to machine code, in contrast to interpreted code
  High-Level such a C -> [Compiler] -> Assembly Code -> [Assembler] -> Machine Code

Instruction Set -- Instruction Types

- We will discuss the various common types of instructions. Though they vary from processor to processor, the similarities make learning one after another fairly painless.
- Common Instruction Types:

1) Data transfer	transfer and store data
2) Control Flow	Make decisions (based on provided operands or based on "side effect " from another operation (type 1 or 2) or based on a status register byte or bit)
3) Arithmetic and Logic	Operate on data

- The source and destination can be a
  - register,
  - a memory location,
  - or an input or output port
- Addressing mode determines how to interpret the value provided by the operands. The addressing mode can be specified by the opcode (different op-code for every address mode) or specified by a prepending data field in the operands
  - Immediate - data is provided in instruction or operand is the data
    - "add a 4" ("a=a+4") : 4 is the immediate data provided in the assembly and machine code
  - Direct - operand provided by the address in memory of data where the data is stored (or to be stored) instead of the data itself
    - (updated) I write your grade on board in a room. You want the data. I provide you the room number. The room number is like the direct operand.
  - Indirect - operand provides address in memory of the address in memory where the data is stored (or to be stored) in memory
    - (updated) In the previous example, I instead provide you a web page address where the room number will be posted.
    - (updated) With indirect, the location of the information (the grade) itself can change,

but you still start in the same place to find it. In the previous example, I can move the grades to another room and change the webpage.

- Register direct - operand indicates which register the data is in or which register the result should be stored in
- Register indirect - operand indicated which register contains the source/destination address of the data in memory
- Indexed Mode - like indirect but address spilt into base and offset and are provided by two separate operands. This is convenient for working with (large) data arrays.
- Program counter relative mode - program counter relative addressing is typically used for flow controls (jumps/branches). Operand is typically signed is added to the P.C. to generate a new instruction address

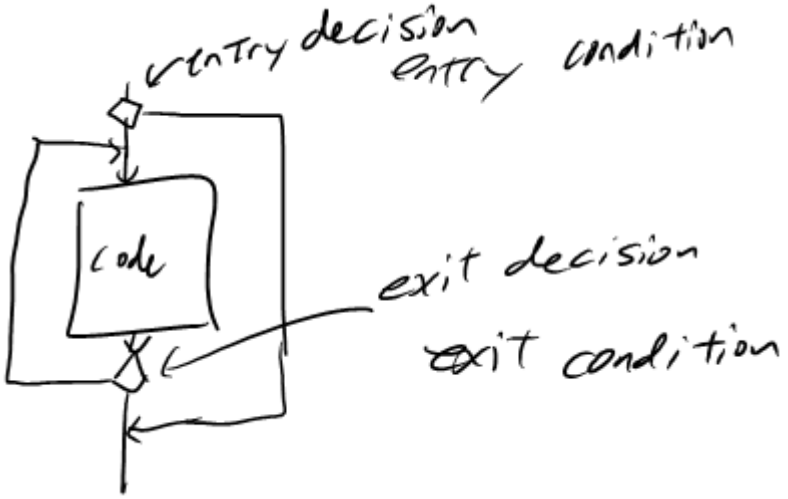
- Data Transfer
  - These instructions require data or location of data. They move or modify memory.
    - Typical Types:
      - Load, LD - move data from memory to register
      - Store, ST - move data from register to memory
      - STI, LDI - store immediate value to memory or register
      - MOVE - move data between registers or between memory locations (really a copy)
      - Exchange, XCH - exchange data in operands
      - PUSH/POP - stack manipulation
      - IN/OUT - transfer data to or from an input/output port
- Execution Flow and Flow Control Instructions
  - Flow concepts:
    - Sequential: default describes using a series of instructions with PC auto incrementing by some amount based on the instruction length/size
    - Branch: decision-based flow: if, else, switch, case, etc..
      - P.C. modified based on operands or contents of flag/status registers (or bits of). Includes some type of test
      - Common Tests

E,NE	Equality, inequality
Z,NZ	Zero, not zero
GT,GE	>,>=
LT,LE	<,<=
V	overflow
C,NC	Cary, no carry
N	negative

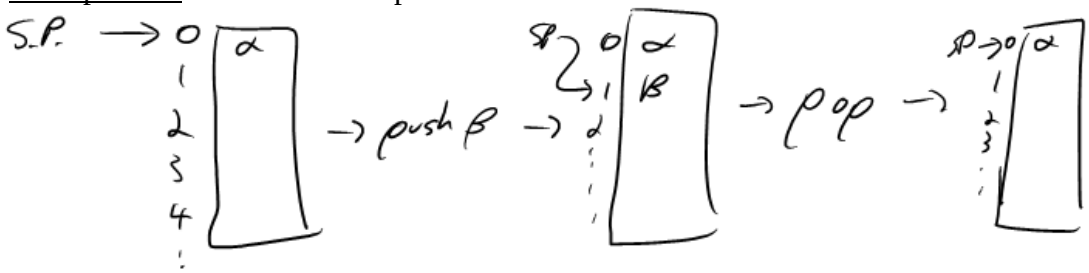
      - Typical branching instructions:
        - Unconditional:
          - BR addess
        - Conditional:
          - BE adddress BNE address
          - also BZ, BNZ, BGT,etc….
      - Jumping can be
        - Absolute, PC = something (operand size must support entire program address space)
        - Relative, PC = PC + something (supports jumps with smaller operands)
      - If-else example

```
dec r0
bz _if_label_
```

```
    ldi r1,0
    br _endif_label_
_if_label_:
    ldi r1, 10
_endif_label_:
    ...
```

- Loop - construct used to run a block of code repeatedly
  - Structure: Entry Decision/Condition, Exit decision/condition, loop body code
  - 

Types of Loops

- Repeat - repeat predetermined number of times
- While - entry decision and exit decision (in code implementation, both may be coded in the same line)
- Do, do While - no entry decision, runs at least once
- For - uses explicit iteration variable that can be used in the code
- Stacks and Procedures / Functions
  - I expect everyone knows stack, top of stack, push, pop
  - At the low-level, stacks are often implemented in a allocated memory array/block with a stack pointer that tracks the top of the stack
  - 

Procedures and Functions

- Use stack in memory
- Process of a function or procedure call (refer to text for complete descriptions)
  - Push next instruction address into stack or store in memory so we can return to the point just after the call in the code
  - Push parameters
  - Set PC to start of function/procedure code
  - Pop parameters
  - if returning values using stack, must pop instruction address and save in

register/memory since return values will bury the return address
        - Do procedure/function work
        - Push return values if function
        - Set PC based on saved instruction address if a or pop off stack
        - Continue execution after original calling point
    - Example on Page 30
    - Supported through CALL and RET assembly statements

Arithmetic and Logic Operations

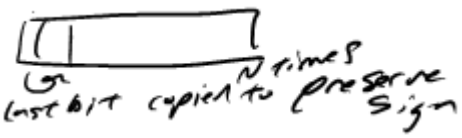
- Arithmetic Operations
- add, sub, mul, div
- addc - add with carry also adds carry bit, supporting long additions)
- subb - subtract with borrow uses status register/bit to support long subtractions)
- inc,dec
- test - operand(s) tested and results affect status flags but result not saved to registers

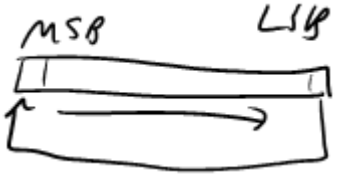
Comparison Instructions in Intel ASM

test arg2, arg1 - Performs a bit-wise AND on the two operands and sets the flags, but does not store a result.

cmp arg2, arg1 - Performs a subtraction between the two operands and sets the flags, but does not store a result.

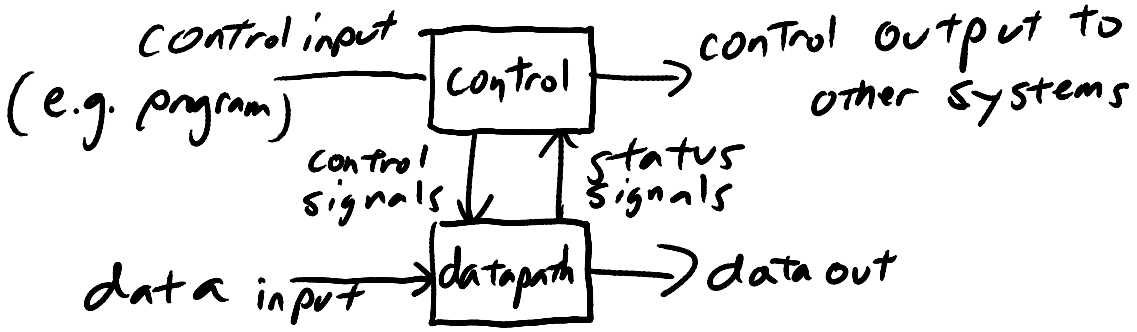
- testset - atomic test and set (will learn about atomic operations later)

- Logic Operations
    - Bitwise operations , e.g. and,or,xor, not or inv
    - set and clear bits in registers, e.g. CLR, SET
    - set and clear bits in status registers, e.g. CLRC, SETC
- Shift Operations:
    - Types: logical, arithmetic (sometimes called a signed shift), rotate
    - Logical
        - SHR
            - Fills on left with 0's, discards bits shifted out on right
            - 
        - SHL
            - Fills on right with 0's, discards bits shifted out on left
    - Arithmetic
        - May affect some flags like overflow and underflow
        - SHRA
            - Fills on left with copies of original sign, preserving the sign of the operation. Discards bits shifted out on right
            - 
            - Can almost mimick integer divide by power of two except result is rounded towards negative infinity instead of 0, so it is the same for positive numbers but slightly different for negative numbers (-1 shifted right arithmetically by 1 is -1 instead of 0)
        - SHLA
            - same as SHL but may affect status flags
            - can mimick mult by power of two

- Rotate
  - No bits discards, instead they are used to fill other end
  - ROR
    - 

  - ROL

Most digital systems fit into this framework:



- Hardware circuit implementing processor must support fetch, decode, execute, next cycle of code execution and all supported instructions
  - Note, the number of clock cycles required per instruction depends on processor hardware and the operation being performed

A microprocessor:

- Components and interconnections must support the instruction cycle
  - Fetch
  - Decode
  - Execute (compute, set)
  - Next (set next instruction address)
- Instruction set determined and limited by hardware design
  - Size and number of decoders, buses, arithmetic circuits, registers, etc. all relate to the supported instruction set