CPU I

Overview

Chapter 4

CPU

- CPU components
  - General
    - Processor : ALU
    - Memory : code and data
    - IO
  - Instructions
    - Opcodes and oprands
    - Addressing

Embedded System Architecture

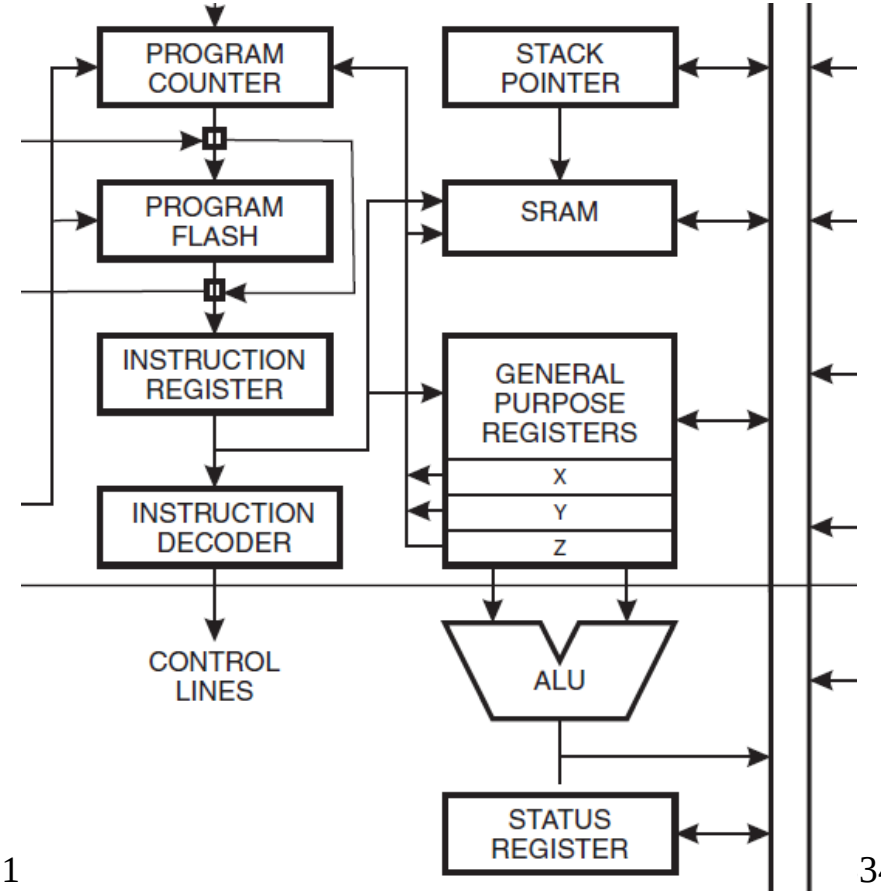
- von Neumann
  - A processing unit
  - A memory to hold both instructions and data
  - CPU can either read an instruction or access data from the memory, but not at the same time
- Harvard
  - A processing unit
  - Two memories to hold instructions and data separately
  - CPU can both read an instruction and access data at the same time

IO Architecture

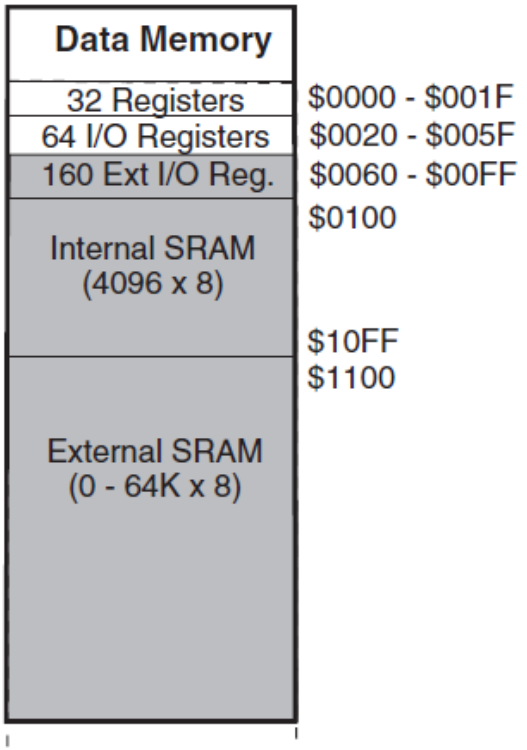
- Port IO
  - IO address is not memory address
  - IO data path is separated from memory data path
  - Special IO instructions
  - Simultaneous access to IO and memory
- Memory mapped IO
  - IO address is a part of memory address
  - IO and memory share the same data path
  - No special IO instructions
  - Exclusive access to IO or memory

Example: ATmega128

- CPU architecture
  - Harvard



- IO architecture
  - Both port and memory IO spaces



Master Processor

- Integrated processor
  - Smaller IO latency
  - Simpler board layout
  - Less power consumption
- Two categories
  - Microprocessor
    - Minimum IO components and on-chip memory
    - Specialized for processing (computing)
  - Microcontroller
    - Maximum IO components and on-chip memory
    - Specialized for controlling

Power Management

- Enery consumption (CMOS property)
- Dynamic Voltage Scaling (DVS)
  - $E = C \times V^2$
  - $f_{max} = F \times (1 - \frac{c}{V})$
- Example
  - V is in range of 1.8V-3.3V, and C is 2.75 W/V^2
  - F is 3.8GHz, and c is 1.563V
  - If V is 3.3V, E is 29.95W, and f is 2GHz
  - If V is 1.8V, E is 8.91W, and f is 500MHz

ACPI

- Advanced Configuration and Power Interface
- Global states
  - G0 (S0): on
  - G1: sleeping
    - S1: CPU stops executing instructions, but power to CPU and memory is on. Other devices may be powered down.
    - S2: CPU is powered off
    - S3: Standby. Everything goes off except memory and what can wake up the computer.
    - S4: Hibernation. Data in memory is saved to non-volatile memory such as a hard drive, and is powered down.
  - G2 (S5): soft off: Everything goes off except what can wake up computer, such as keyboard, NIC, USB.
  - G3: shutdown

ACPI

- Processor states:
  - C0: on
  - C1: halt. Processor is not executing instructions, but can return to an executing state essentially instantaneously.
  - C2: stop-clock. Processor maintains all software-visible state, but may take longer to wake up.
  - C3: sleep. Processor does not need to keep its cache coherent, but maintains other state.

ACPI

- Device states
  - D0: on
  - D1: less power consumption (e.g. slower)
  - D2: even less power consumption (e.g. partially off)
  - D3: off
- Performance states
  - P0: max power and frequency
  - Pn: less than P(n-1), voltage/frequency scaled

Example: Atmega128

- Instruction: sleep
- MCU Control Register
  - SE: sleep enable
  - SM2, SM1, and SM0: sleep modes
    - Idle
    - ADC noise reduction
    - Power-down
    - Power-save
    - Standby
    - Extended standby

Example: Atmega128

- Example: Enable sleep
  - How to configure the sleep control register
  - The meaning of different sleep modes
- Example: hello.avr.asm
  - How it handle return from main
  - How to sleep

Instruction Set Architecture

- ISA
  - Operation / opcode / instruction
    - The format and types of instructions
  - Operand
    - Data in instructions
  - Storage
    - How instructions and data are stored
  - Addressing mode
    - How to access memory
  - Interrupt handling
    - Handling of hardware and software interrupts

ISA

- Functionality
  - What can the instructions do?
- Performance
  - How fast can a program be executed?
  - How much power can a program consume?
- Design time
  - How much time is needed to implement?
- Cost
  - How much do a processor and peripherals cost?

Opcodes

- Types
  - Computation
  - Movement
  - Branch
  - I/O
  - Context switch
- Formats in machine
  - Fixed size
  - Variant
- A for loop
  - for (i=0; i<10; i++)

Operands

- Types
  - Bits, smaller than one byte
  - Byte, 8 bits
  - Halfword, 16 bits
  - Word, 32 bits
- Interpretation
  - Integer
  - Character
  - Floating point
- Formats in assembly
  - Binary
  - Decimal
  - Hex

Operands

- High level: nesc
  - int16_t
  - uint16_t
  - int8_t
  - uint8_t
  - char
  - * : void*, char*, uint8_t*, int16_t*
- Low level: avr
  - constant
  - register
  - memory address

Storage

- Organization of address space
  - Linear
  - Segmented
- Byte order (not bit order in a byte)
  - Big-endian : MSB in lower address, 68000
  - Little-endian : MSB in higher address, x86
  - Configurable : ARM, MIPS, PowerPC

Byte Ordering

- Little-endian : LSB in lower address

```
int foo=0x0A0B0C0D;
*(((char*)&foo)+2)=?

struct _foo {
  char a='A';
  int b=0x0A0B0C0D;
} foo;
*(((char*)&foo)+2)=?
```

Storage

- Register set
  - Fast programmable memory used to store operands that are immediately or frequently used.
- Types
  - General purpose
  - I/O
  - Program counter
  - Status
  - Stack pointer

Storage

- Example: registers
  - Status register
  - General purpose registers
    - X, Y, Z
  - Others
    - Stack pointer
    - IO registers

Bit	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

7 0	Addr.	
R0	$00	
R1	$01	
R2	$02	
...		
R13	$0D	
R14	$0E	
R15	$0F	
R16	$10	
R17	$11	
...		
R26	$1A	X-register Low Byte
R27	$1B	X-register High Byte
R28	$1C	Y-register Low Byte
R29	$1D	Y-register High Byte
R30	$1E	Z-register Low Byte
R31	$1F	Z-register High Byte

Addressing Modes

- Hypothetical instructions.
- Implied mode : push
- Immediate mode : ldi r1, 0x10
- Register modes : ldi r1, 0x10
- Direct mode : lds r1, 0x10
- Indirect mode : lds r1, (0x10)
- Register indirect modes : lds r1, (r2)
- Relative mode : rjmp 0x10
- Indexed mode : ldx r1, (r2)[4]

Data Addressing Modes

- Register direct
  - Single register Rd
  - Two registers Rd and Rr
- IO direct
- Data direct
- Data indirect with displacement
- Data indirect
- Data indirect with pre-decrement
- Data indirect with post-decrement

Program Addressing Modes

- Program memory constant addressing
  - LPM, ELPM, SPM
- Program memory with post-increment
  - LPM, ELPM, SPM
- Direct program addressing
  - JMP, CALL
- Indirect program addressing
  - IJMP, ICALL
- Relative program addressing
  - RJMP, RCALL

Interrupt Handling

- Interrupt
  - Mechanism to stop the current flow of the program in order to execute another set of code in response to some event.
- Events
  - Reset
  - I/O
  - Timer
  - Computation error
  - Execution error

ISA Model

- Application-specific ISA models
  - Designed for specific embedded applications
- General purpose ISA models
  - Designed to be used in various systems
- Instructional-level Parallelism ISA models
  - Designed to execute instructions in parallel

Application-specific ISA Models

- Controller model
  - No complex data manipulation
  - Audio and video sampling I/O controller (ASIC)
- Datapath model
  - Repeatedly perform fixed computations
  - Fourier transformation in DSP
  - Multimedia computation in GPU
  - Float point computation in FPU
- FSM with datapath model
  - ASIC + DSP
- JVM model
  - JME application-specific professor

General-purpose ISA Models

- CISC model
  - Complex instructions of several low-level operations
  - E.g., load + compute + store
  - Intel x86, Motorola 68000
- RISC model
  - Simple instructions of reduced low-level operations
  - Single cycle per low-level operation / instruction
  - ARM, SPARC, MIPS

Instruction-level Parallelism ISA Models

- Single instruction multiple data model
  - Data level parallelism : scalar vs. vector
  - Multimedia computation: Intel MMX
    - E.g., brightness adjust
  - SISD, SIMD, MISD, MIMD
- Superscalar machine model
  - Instruction-level parallelism : scalar vs. superscalar
  - Multiple functional units (ALUs)
  - PowerPC 970: 4 ALUs, 2 FPUs, and 2 SIMD units
- Very long instruction word computing model
  - One instruction encodes multiple operations
  - Each operation uses one functional unit of the processor
  - Compiler decides the instructions : VLIW vs. superscalar
    - E.g., f12=f0*f4, f8=f8+f12, f0=dm(i0,m3)