CSSE2010

CSSE2010/CSSE7201
Lecture 12

Instruction Set Architecture – Part 1

School of Information Technology and Electrical Engineering
The University of Queensland

Instruction Set Architecture (ISA)

- **ISA** defines the interface between hardware and software
  - ISA is a specification
  - **Microarchitecture** is how the control unit is built
- For hardware (microarchitecture) designers
  - Don’t need to know about the high level software
  - Just build a microarchitecture that implements the ISA
- For software writers (machine language programmers and compiler writers)
  - Don’t need to know (much) about microarchitecture
  - Just write or generate instructions that match the ISA

What makes an ISA?

1. Memory models
2. Registers
3. Instructions
4. Data types

What makes an ISA?
1: Memory Models

- Memory model: how does memory look to CPU?
- Issues
  A. Addressable cell size
  B. Alignment
  C. Address spaces
  D. Endianness

A. Addressable Cell Size

- Memory has cells, each of which has an unique address (or cell number).
- Most common cell size is 8 bits (1 byte).
  - But not always!
    - AVR Instruction memory has 16 bit cells
- Bus is used to transport the content of cell, but sometimes the data bus may be wider:
  - i.e. retrieve several cells (addresses) at once
    - Address bus does not need to be as wide

Bus sizes

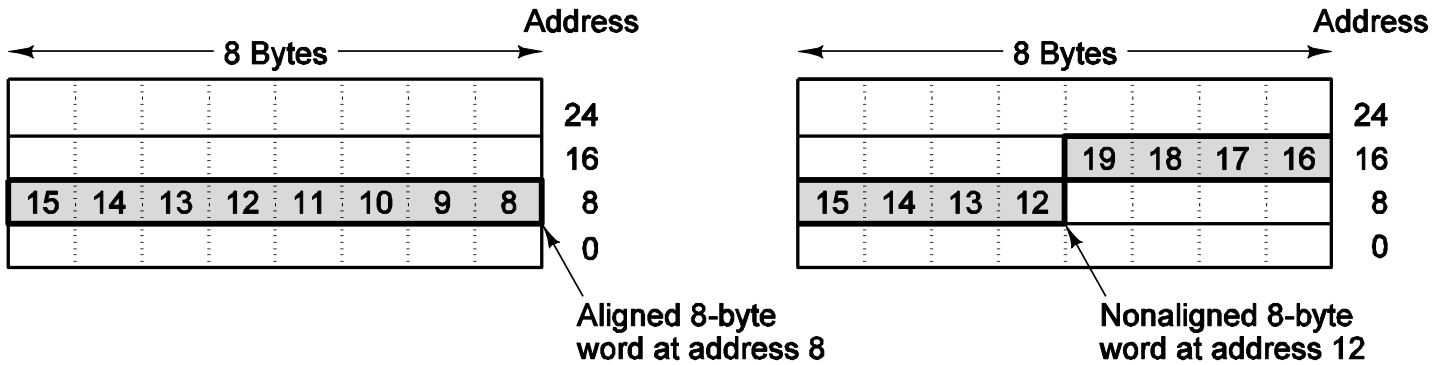
- For every doubling of data bus width
  - Remove least significant bit of address *bus*
- Example: 2^n cells x 8-bit cell size
  - Address size: n-bits
- Bus configurations:
  - Data bus: 8 bits Address bus: n bits
    - One cell transferred at a time
  - Data bus: 16 bits Address bus: n-1 bits
    - Two cells transferred at a time
  - Data bus: 32 bits Address bus: n-2 bits
    - Four cells transferred at a time

If a CPU supports 2^{35} bytes of memory and has a 64 bit data bus, how many bits wide is the address bus?

10

B. Alignment

- Many architectures require **natural alignment**, e.g.
  - 4-byte words starting at addresses 0, 4, 8, …
  - 8-byte words starting at addresses 0, 8, 16, …
- For example, with a 8 byte word, it can be stored as follows:



B. Alignment (cont.)

- Alignment often required because it is more efficient
- Example – Pentium IV
  - Fetches 8 bytes at a time from memory (64-bit wide data bus)
  - Alignment is NOT required (so the processor is backwards compatible with earlier processors)
    - e.g. 4-byte word stored at address 4 takes 1 read (bytes 0 to 7).
    - e.g. 4-byte word stored at address 6: Must read bytes 0 to 7 (one read) and bytes 8 to 15 (second read) then extract the 4 required bytes from the 16 bytes read

How many memory reads are needed to read an 8-byte value on a machine with a 16-bit data bus that does not require natural alignment?

0% A. Always 2

0% B. 2 or 3

0% C. Always 3

0% D. 3 or 4

0% E. Always 4

0% F. 4 or 5

0% G. Always 5

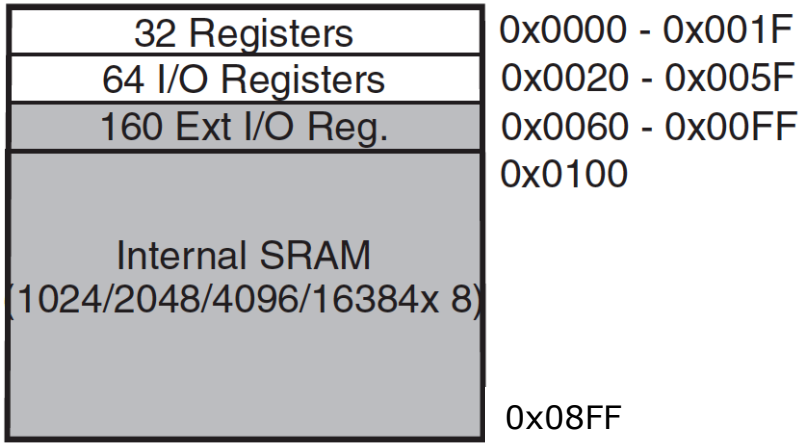
10

C. Address Spaces

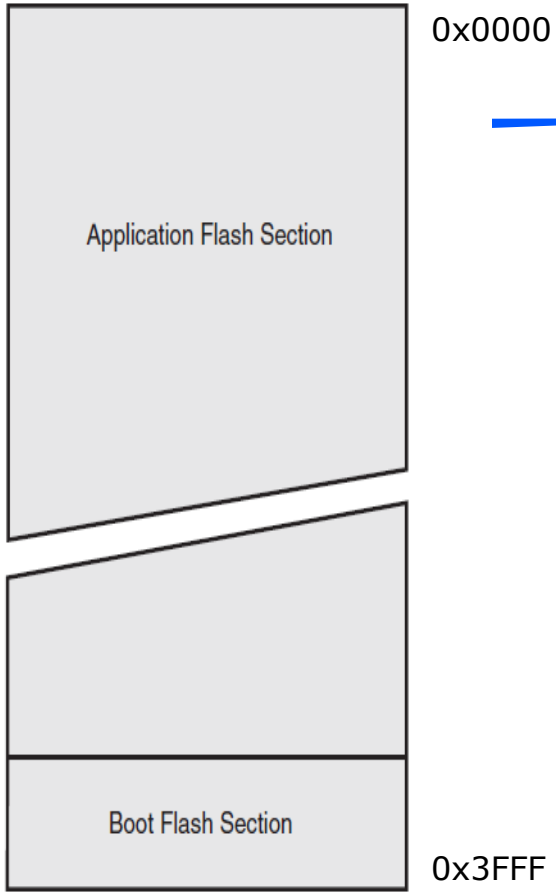
- Many microprocessors have a single linear memory address space (**von Neumann** architecture)
  - e.g. 2^{32} bytes numbered from 0 to 2^{32} -1
    - (may not be bytes – depends on addressable cell size)
- However, **Harvard architecture** is:
  - Separate address spaces for instructions and data
  - AVR ATmega 324A
    - Data address space: 2^{11} (2048) bytes
    - Instruction address space: 2^{14} 16-bit words
      - 16,384 instruction words = 32,768 bytes

AVR Address Spaces

Data



Program Memory



[From ATmega324A datasheet]

D. Endianness

- Different machines may support different byte orderings
- **Little endian** – little end (least significant byte) stored first (at lowest address)
  - Intel microprocessors (Pentium etc)
- **Big endian** – big end stored first
  - SPARC, Freescale (formerly Motorola) microprocessors
- Most desktop/workstation/server CPUs produced since ~1992 are "bi-endian" (support both)
  - some switchable at boot time
  - others at run time (i.e. can change dynamically)

Example

- The contents of a memory are shown below. What is the value of:
  - the 1-byte word at address 1?
  - the 2-byte word at address 0 if this is a big endian system?

0	00001111
1	00110011
2	11011101
3	00101100

The contents of a memory are shown below. What is the value of the 2-byte word at address 2 if this is a little endian system?

0% A. 00110011 11011101

0% B. 11011101 00110011

0% C. 11011101 00101100

0% D. 00101100 11011101

0	00001111
1	00110011
2	11011101
3	00101100

10

What makes an ISA?

1. Memory models
2. Registers
3. Instructions
4. Data types

What makes an ISA? 2: Registers

- Two types of register
  - General purpose
    - Used for temporary results etc
  - Special purpose, e.g.
    - Program Counter (PC)
    - Stack pointer (SP)
    - Input/Output Registers
    - Status Register
      - (Tanenbaum calls this Program Status Word)

2: Registers (cont.)

- Some other registers are part of the microarchitecture NOT the ISA
  - e.g. Instruction Register (IR)
  - i.e. programmer doesn't need to know about these (and can't directly change or use them)

AVR Registers

- General purpose registers (0-31) are quite regular and can be used interchangeably
  - Exception: a few instructions work on half the registers (registers 16-31)
    - (Only 4 bits used to represent operand in instruction)
  - X, Y, Z registers
    - Next slide
- There are many I/O registers
  - Not to be confused with general purpose registers
  - Some instructions work with these, others with general purpose registers – don't confuse them

AVR X,Y,Z registers

R0	0x00	
R1	0x01	
R2	0x02	
…		
R13	0x0D	
R14	0x0E	
R15	0x0F	
R16	0x10	
R17	0x11	
…		
R26	0x1A	X-register Low Byte
R27	0x1B	X-register High Byte
R28	0x1C	Y-register Low Byte
R29	0x1D	Y-register High Byte
R30	0x1E	Z-register Low Byte
R31	0x1F	Z-register High Byte

AVR I/O Registers

- AVR ATmega324A has 224 I/O register addresses to control peripherals and get data to/from them, e.g.
  - Timers and counters
  - Analog to Digital Converters
  - Serial input/output
  - General purpose input/output ports
  - Three registers associated with each (learning lab 11)
    - **DDRx** - Data direction register (x = A,B,C or D)
    - **PORTx** - Values to output
    - **PINx** - Values on the pins
- Some addresses reserved (not actually available on this device)

What makes an ISA?
3: Instructions

- This is the main feature of an ISA
- Instruction types include
  - **Input/Output** – communicate with I/O devices
  - **Load/Store** – move data from/to memory
  - **Move** – copy data between registers
  - **Arithmetic** – addition, subtraction, …
  - **Logical** – Boolean operations
  - **Branching** – for deciding which instruction to perform next

Recall - AVR Instructions

- **`MOV rd, rr`**
  - "move" (but actually means copy) contents of register rr (source) to register rd (destination)
  - Example: **`mov r3, r14`**
- **`ADD rd, rr`**
  - Add contents of register rr to register rd
  - Example: **`add r5,r6`**
- **`AND rd, rr;  OR rd, rr;  EOR rd,rr`**
  - Bitwise operations on given two registers, result goes into rd
- **`LDI rd, K`** (where K is a constant)
  - Load constant value into a register (16 to 31)

Input/Output Instructions

- AVR ATmega324A has 224 I/O registers to control peripherals
- I/O Registers 0 to 223
  - Available at memory addresses 32 to 255 ($20 to $FF)
    - Use load and store instructions to access
- First 64 I/O registers (0 to 63) are special
  - Can also be accessed using "`in`" and "`out`" instructions
- Other 160 I/O registers (64 to 223) are called "extended" I/O registers
  - Can only be accessed using load and store instructions

IN and OUT instructions

- **in *rd, P***
  - Load I/O register value into general purpose register
  - rd: r0 to r31
  - P = I/O register number (0 to 63, 0 to 0x3F)
  - Example:

- **out *P, rr***
  - Store value from general purpose register into I/O register
  - rr: r0 to r31, P = 0 to 63 (0 to 0x3F)
  - Example:

Summary: What makes an ISA?

1. Memory models
2. Registers
3. Instructions (more in lecture 13)
4. Data types (lecture 13)

- If you know all these details, you can
  - Write machine code that runs on the CPU
  - Build the CPU