Politechnika Wrocławska

Computer Architecture and Organization

Lecture 4

Dr. Radosław Michalski

Department of Computational Intelligence, Faculty of Computer Science and Management, Wrocław University of Science and Technology

Version 1.0, spring 2017

Source and licensing

The most current version of this lecture is here:
https://github.com/rmhere/lecture-comp-arch-org

Overview of this lecture

CPU, FPGA, ASIC

MIPS assembly language

CPU, FPGA, ASIC

How FPGA is different from a CPU?

- **f**ield-programmable **g**ate **a**rray - **FPGA**
- an idea that started in 1980's
- goal: to build a reprogrammable chip
- no fixed gates' configurations
- more flexible architecture
- hardware description language (HDL) for specifying the configuration

CPU, FPGA, ASIC

FPGA applications

- previously: for low-volume applications
- now: almost everywhere
  - consumer electronics
  - medical applications
  - video and image processing
  - automotive
  - many, many more

CPU, FPGA, ASIC

ASIC

- **a**pplication-**s**pecific **i**ntegrated **c**ircuit - **ASIC**
- specific task to be performed
- the purpose has to be known before manufacturing
- fixed configuration (security)
- applications
  - aerospace (more tolerant to radiation)
  - DSP
  - cryptography

CPU, FPGA, ASIC

Comparison

- **CPU**: most expensive, high power demand, fixed HW configuration, general purpose
- **FPGA**: cheaper, high number of gates, reprogrammable logic, more flexible, also for prototyping ASICs
- **ASIC**: cheapest, specific purpose, fixed HW configuration, power saving

CPU, FPGA, ASIC

Comparison - mining Bitcoins

Satoshi, public domain

bitcoinwiki - Non-specialized hardware comparison
bitcoinwiki - Mining hardware comparison

CPU, FPGA, ASIC

Comparing chip types

Video

Amplex Electrosystem - CPU vs FPGA vs ASIC

MIPS assembly language

Program structure overview

- plain text file
- data declaration first
- program code section later

MIPS assembly language

Program structure

Data declarations:

- followed by the assembler directive `.data`
- declare variable names to be used in program
- these are stored in main memory (RAM)

Code:

- followed by the assembler directive `.text`
- first instruction run after `main:`

Comments:

- `# starts a comment`

MIPS assembly language

Program structure - example

```
# Comment giving name of program and description of function
# Template.s
# Bare-bones outline of MIPS assembly language program

    .data           # variable declarations follow this line
                    #
    .text           # instructions follow this line
   main:            # indicates start of code

# End of program, leave a blank line afterwards.
```

MIPS assembly language

Assembly instructions

- each line contains one instruction (operation)
- fixed order and number of operands
- more operands needed in high level language?
  - more instructions needed in assembly

MIPS assembly language

Assembly operands

- no variables, just registers (load and store)
- data declarations refer to memory

MIPS assembly language

Bytes and words

- byte - 8 bits
- word - 4 bytes - 32 bits

MIPS assembly language

Data declaration

Format:

```
name:    storage type    value(s)
```

Example:

```
var1:    .word     3         # integer
array1:  .byte     'a','b'   # a 2-element char array
array2:  .space    40        # 40 consecutive bytes
```

MIPS assembly language

Registers

Number	Name	Description
\$0	\$zero	Always zero
\$1	\$at	Reserved for assembler
\$2-\$3	\$v0, \$v1	First and second return values, respectively
\$4-\$7	\$a0...\$a3	First four arguments to functions
\$8-\$15	\$t0...\$t7	Temporary registers
\$16-\$23	\$s0...\$s7	Saved registers
\$24-\$25	\$t8, \$t9	More temporary registers
\$26-\$27	\$k0, \$k1	Reserved for kernel (operating system)
\$28	\$gp	Global pointer
\$29	\$sp	Stack pointer
\$30	\$fp	Frame pointer
\$31	\$ra	Return address

MIPS assembly language

System calls

- used to read or print values or strings from input/output window, and indicate program end
- use syscall operating system routine call
- first supply appropriate values in registers \$v0, \$a0-\$a1, \$f
- result value (if any) returned in register \$v0

MIPS assembly language

System calls - list

Service	Code in $v0	Arguments	Results
print_int	1	$a0 = integer to be printed	
print_float	2	$f12 = float to be printed	
print_double	3	$f12 = double to be printed	
print_string	4	$a0 = address of string in memory	
read_int	5		integer returned in $v0
read_float	6		float returned in $v0
read_double	7		double returned in $v0
read_string	8	$a0 = memory address of string input buffer, $a1 = length of string buffer (n)	
sbrk	9	$a0 = amount	address in $v0
exit	10		

MIPS assembly language

An example of assembly code - goal

Write a program that computes the following expression:

`(a + b) - (c - d)`

MIPS assembly language

An example of assembly code - source

```
.data
    a:   .word 5
    b:   .word 6
    c:   .word 4
    d:   .word 3

.text
main:
    lw $t0, a
    lw $t1, b
    lw $t2, c
    lw $t3, d

    add $t4, $t0, $t1
    sub $t5, $t2, $t3
    sub $t6, $t4, $t5

    li $v0, 1
    add $a0, $zero, $t6
    syscall
```

MIPS assembly language

MARS - demo

Demonstration of how the code works in MARS MIPS Simulator.

MIPS assembly language

Sources & additional materials

- D. Reed, MIPS Architecture and Assembly Language Overview, University of Illinois at Chicago, IL, United States (course materials)
- Y. Kreinin, How FPGAs work, and why you'll buy one (article)
- R. Baruch, Bulding a CPU on an FPGA (videos)