Computer Architecture Exercises

Qiyang Hu, Givi Meishvili, Adrian Wälchli

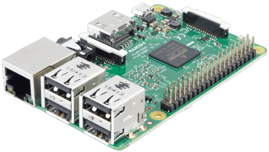
March 27, 2018

Today

Organisational Matters

April 10

- Raspberry Pi's will be distributed to groups
- Deposit: 120 CHF (please bring the exact amount of money)
- Bring your student card
- Sign up as a group in ILIAS
- If you can not be here to pick up the Pi, write an E-Mail to: `waelchli@inf.unibe.ch`



Review of Series 2

Theoretical Part

Exercise 1

Function Pointer

```
void callA(int a) { printf("A: %i\n", a);}
void callB(int a) { printf("B: %i\n", a);}
void callC(int a) { printf("C: %i\n", a);}

void (*functionPointer[3])(int)
        = { &callA, &callB, &callC };

functionPointer[0](20);
functionPointer[1](21);
functionPointer[2](22);
```

Output

A: 20
B: 21
C: 22

Exercise 2

Pointer and const

```
/* regular pointer */
int *  a;
*a = 0; /* OK */          a = 0; /* OK */

/* (variable) pointer to constant */
int const * b;
*b = 0; /* ERROR! */  b = 0; /* OK */

/* constant pointer to variable */
int * const c;
*c = 0; /* OK */          c = 0; /* ERROR! */

/* constant pointer to constant */
int const * const d;
*d = 0; /* ERROR! */  d = 0; /* ERROR! */
```

Exercise 3

Loops and Arrays

```
int array[11] = {1, 0, 1, 0, 1, 0, 1, 0, 1,
0, 1};
int i;

for (i=0; i<=11; ++i) {
  printf("%i ", array[i]);
}
```

Problem: Array has elements $0 \ldots 10$, but loop runs from $0 \ldots 11$
$\Rightarrow$ no check for array boundary

Exercise 4

Assembly in C

```
L1: beq $s2, $s3, L2
    #do something
    add $s2, $s2, $s1
    j L1
L2:
```

```c
int s1 = 1;
int s2 = 4;
int s3 = 20
while (s2 != s3) {
  /* do something */
  s2 = s2 + s1;
}
```

. . . or with `for`-loop

Exercise 5

Expansion of Pseudo-Instructions

```
bge $s2, $s3, Label
```

. . . becomes . . .

```
slt $at, $s2, $s3    #$at set to 1 if $s2 < $s3
beq $at, $zero, Label
```

Exercise 6

Endianness

	Address	**Binary Value**	**LSB**
Word address	10001	0101 1010	Little Endian $\downarrow$
	10002	1011 0110	
	10003	0101 1110	
	10004	1001 1010	Big Endian $\uparrow$
	10005	0110 1001	

$$\begin{aligned}
\text{LE} &= (1001\,1010\,0101\,1110\,1011\,0110\;\underbrace{0101\,1010}_{\text{LSB}})_2 \\
&= (2'589'898'330)_{10} \\
\text{BE} &= (0101\,1010\,1011\,0110\,0101\,1110\;\underbrace{1001\,1010}_{\text{LSB}})_2 \\
&= (1'521'901'210)_{10}
\end{aligned}$$

Exercise 7

Array Access with Assembly (1)

```
void mul(short x[], int index, int mul) {
  x[index] *= mul;
}
```

```
# calculate the array position
sll $t1, $t1, 1
add  $t2, $t2, $t1
# load the array element: 2 Bytes = halfword
lh $t3, 0($t2)
# multiply the array element
mult $t3, $t4
mflo $t3
# store the array element
sh $t3, 0($t2)
```

Exercise 8

Array Access with Assembly (2)

- B Array with ten data words
- Address of B in `s1`
- Load last word of B into `s2` with exactly one instruction

```
#load B[9] from memory, B is stored at $s1
lw $s2, 36($s1)
```

- `sizeof(word)=4`
- Address of $B[9]$ = address of $B + 9 \cdot$ `sizeof(word)`

Optional Questions

Byte Order

- we use Big Endian
- i.e., the address of a word is the address of the Most Significant Byte (MSB)
- e.g. the value

$$0x04\ 03\ 02\ 01 = 0b\underbrace{00000100}_{\text{MSB}}\ 00000011\ 00000010\ 00000001$$

at address `0xAAAA` is stored in memory as

Address	Value
0xAAAA	00000100
0xAAAB	00000011
0xAAAC	00000010
0xAAAD	00000001

Declarations

Listing 1: mips.h

```
extern Operation operations[OPERATION_COUNT];
extern Function functions[FUNCTION_COUNT];
```

- The given structures are essentially dispatchers (by use of function pointers) – enriched with additional information
- They provide the connection between the Op-Code and the actual implementation of the operation

Sign Extension

- Sign extension: When converting a 16-bit number in Two's-Complement to the "equivalent" 32-bit number
- `0x0002` becomes `0x00000002`
- `0xFFFD` $= (-2)_{10}$ becomes `0xFFFFFFFD` $= (-2)_{10}$
- but not `0x0000FFFD` $= (65533)_{10}$

Programming Part

storeWord

```
/* Store a word to memory */
void storeWord(word w, word location) {
        memory[location]   = (w >> (8*3)) & 0xFF;
        memory[location+1] = (w >> (8*2)) & 0xFF;
        memory[location+2] = (w >> (8*1)) & 0xFF;
        memory[location+3] =  w           & 0xFF;
}
```

add

Specification of `add` (Format: R)

```
R[rd] = R[rs] + R[rt]
```

Implementation:

```
/* ADD */
void mips_add(Instruction *instruction) {
        InstructionTypeR r = instruction->r;
        registers[r.rd] = (signed)registers[r.
           rs] + (signed)registers[r.rt];
}
```

addi

Specification of `addi` (Format: I)

```
R[rt] = R[rs] + SignExtImm
```

where

```
SignExtImm = { 16{immediate[15]}, immediate }
```

Implementation:

```
/* ADDI */
void mips_addi(Instruction *instruction) {
        InstructionTypeI i = instruction->i;
        registers[i.rt] = (signed)registers[i.
           rs] + (signed)signExtend(i.
           immediate);
}
```

jal

Specification of `jal` (Format: J)

```
R[31] = PC + 4; PC=JumpAddr
```

where

```
JumpAddr = { PC[31:28], address, 2'b0 }
```

Implementation:

```c
/* JAL */
void mips_jal(Instruction *instruction) {
  RA = pc;
  pc = (pc & 0xF0000000) | (instruction->j.
      address << 2);
}
```

lui

Specification of `lui` (Format: I)

```
R[rt] = {imm, 16'b0 }
```

Implementation:

```
/* LUI */
void mips_lui(Instruction *instruction) {
        InstructionTypeI i = instruction->i;
        registers[i.rt] = i.immediate << 16;
}
```

sw

Specification of sw (Format: I)

```
M[R[rs]+SignExtImm] = R[rt]
```

where

```
SignExtImm = { 16{immediate[15]}, immediate }
```

Implementation:

```
/* SW */
void mips_sw(Instruction *instruction) {
        InstructionTypeI i = instruction->i;
        storeWord(registers[i.rt], registers[i
            .rs] + (signed)signExtend(i.
            immediate));
}
```

Introduction to Series 3

- Last exercise about C programming
- Read carefully, think, then start programming
- Parameters for `main` program
- Working with files
- Functions with variable number of arguments
- **Deadline: April 17, 15:00**
- As usual, upload everything to ILIAS: PDF & zip-file