Architecture of Digital Systems I
WS 2017/2018

Technische Universität Kaiserslautern
Fachbereich Elektrotechnik und Informationstechnik
Entwurf informationstechnischer Systeme

Assignment 1

Problem 1.1

For each of the following keywords find the description from the list below that matches best. Each description should be used only once.

a) Interpreter
b) Assembler
c) Compiler
d) Bit
e) CPU
f) Data path
g) Control unit
h) Main memory
i) Cache
j) Register
k) RISC
l) Operating system

1. Program translating a symbolic representation of a machine instruction into a binary representation.
2. Binary digit
3. Another name for processor
4. Component of a processor performing arithmetic operations.
5. Component of a processor controlling all other components.
6. Program that executes instructions written in a high level programming language.
7. Fast buffer memory with short access time.
8. Program that translates a high level language program into assembler language.
9. Place where data is stored that will immediately be processed further on.
10. Processor concept with regular instruction formats and regular execution times
11. Program controlling access of other programs to resources (e.g. memory, CPU).
12. Place where programs to be executed are stored

Problem 1.2

Describe with a few words what happens inside a processor when an addition instruction is executed.

Problem 1.3

Consider the following MIPS assembler code. Assume that register `a0` is used for the input and initially contains a positive number n. Assume that register `v0` is used for the output.

Register `zero` is the so-called zero-register; a read-only register that always contains '0'. The instruction `slt $1,$2,$3` (set-on-less-than) assigns register `$1` with the value '1' when `$2` < `$3`, and assigns it '0' if not.

```
begin:  addi  $s0,$zero,0
        addi  $s1,$zero,1
loop:   slt   $t0,$a0,$s1
        bne   $t0,$zero,finish
        add   $s0,$s0,$s1
        addi  $s1,$s1,2
        j     loop
finish: add   $v0,$s0,$zero
```

a) Add comments to every line of the program.

b) Write a C code segment that a compiler would translate into this assembler code.

c) Describe in a single sentence what the program computes.

Problem 1.4

a) Translate the following C-code fragment into a sequence of MIPS assembly language instructions. The variables `a`, `b` and `c` are arrays of integer values. The number of elements in all three arrays is contained in `size`.

Assume that register `s4` contains size and that the registers `s0`, `s1` and `s2` contain the addresses of `a`, `b` and `c` respectively. (Any free register, for example `s3`, can be used for `n`.)

```
for(int n=0; n<size; ++n)
{
   c[n]=a[n]+b[n];
}
```

b) A computer operating at a clock frequency of 2 GHz executes the program. The number of clock cycles used to execute each type of instruction is shown in the following table.

Instruction	#clock cycles
add, addi, slt	4
lw, sw, beq, bne, j	5

Assuming a value of 1000 for `size`, how much time is required to execute the program?

Problem 1.5

Consider following MIPS code:

```
1        bne $s1,$s2,L1
2        ...
3  L1:   add $s3,$s4,$s5
```

Assume that the difference between the memory addresses of the two shown instructions is 2^{23}. Can this program be translated into a binary representation without further modifications? If not, how can the program be modified to solve the problem?

Problem 1.6

Write a procedure `convert` in MIPS-Assembler language that converts a null-terminated character string to its corresponding integer number. (Characters are ascii coded http://www.asciitable.com.)

The first characters of the string contains the most significant digits. The procedure expects its address in register `$a0`. The trailing null-character is used to mark the end of the string. The result should be found in register `$v0` when the procedure returns. The procedure should return `-1` if the string contains one or more "non-digit" characters. Note:

- It might be easier to write a C-Code first and then manually compile it to assembly language.
- For loading a single byte you may use the instruction `lbu` (load byte unsigned):
  - `lbu $1,x($2)`: The byte found in memory cell (`$2` + x) is copied into the lower 8 bits of `$1`. The remaining bits of `$1` are set to 0.
- Logical shift operations may be performed by using the instructions:
  - `sll $1,$2,x`: shift `$2` left logically by x digits and store result in `$1`
  - `sllv $1,$2,$3`: shift `$2` left logically by `$3` digits and store result in `$1`

- In a logical shift all bits are shifted (also the sign bit). Empty bits are filled with 0's. Bits shifted out are ignored (no overflow). E.g., "shift one left" of "1111" is "1110".
- Shift operations, like logic and arithmetic shift, will be introduced thoroughly in chapter 4 (Computer Arithmetic) of the lecture.

Problem 1.7

Write a procedure called `zerofind` in MIPS assembler language. The procedure expects the start address of an integer array in register `$a0` and the address of the last element of the array in register `$a1` (for an empty array we have `$a1` < `$a0`). The procedure should locate the first zero in the array and return its address in register `v0`. If there are no zeros in the array, `zerofind` should return the value zero.

Problem 1.8

Write a procedure called `zerocount` in MIPS assembler language that returns (in `$v0`) the number of zeros in an integer array. Like `zerofind` from the previous exercise, the start and end address of the array are stored in `$a0` and `$a1`, respectively.

The procedure should call `zerofind`, and, instead of using loops, it should be implemented as a recursive procedure.