Institute of Computer Science

Prof. Dr. Armin Biere
Dr. Mathias Fleury

Freiburg, 10/11/2021

Computer Architecture
Exercise Sheet 2 (version 2022-8)

Exercise 1

In this exercise, your task is to transfer C code statements into RISC-V assembly instructions and vice versa. Figure 1 (page 4) shows the RISC-V instructions. For every part of the exercise assume that the C variables f, g, h, i, and j have already been placed in registers $x5$, $x6$, $x7$, $x28$, and $x29$, respectively. If you need to calculate intermediate results, you can use any register not used for the task at hand. But you may not override any of the used variables that may still be used by subsequent commands.

a) For the following C statement, write the corresponding RISC-V assembly code using a minimal number of instructions. Notice that the `addi` instruction can also contain negative operands!

```
f = g + (h - 5);
```

b) Write a single C statement that corresponds to the two RISC-V assembly instructions below.

```
add x5, x6, x7
add x5, x7, x5
```

c) For the following C statement, write the corresponding RISC-V assembly code using *word* instructions. Assume that the base addresses of the arrays A and B are in registers $x10$ and $x11$, respectively. E.g. `lw x5, 40(x10)` would load $A[10]$ into register $x5$.

Notice that a word uses 4 bytes in memory, whereas each memory address refers to a single byte. So you have to multiply the array indexes by 4.

```
B[8] = A[i-j];
```

(Taken from "Computer Organization and Design" by David A. Patterson, John L. Hennessy)

Exercise 2

We consider the well-known Fibonacci sequence

```
int fib (int n)
{
  if (n <= 1)
    return 1;
  else
    return fib(n-1)+fib(n-2);
}
```

a) Write the corresponding RISC-V assembly code.

b) Consider the following version that is equivalent to the previous code:

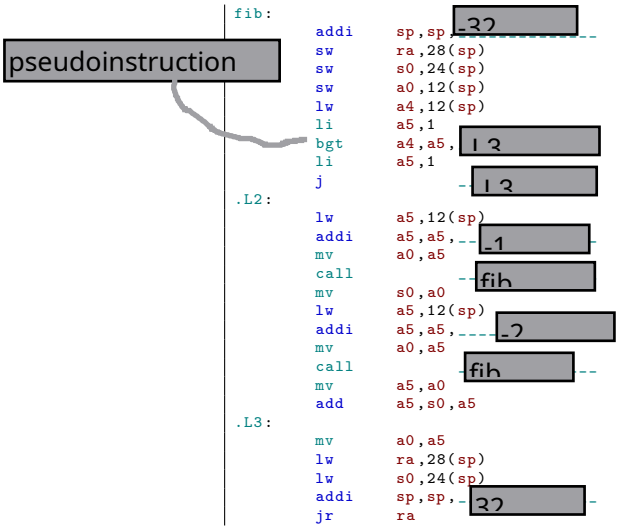
```
int fib (int n)
{
  int a = 1;
  int b = 1;
  int k = 1;
  while (k < n)
  {
    const int c = b;
    b += a;
    a = c;
    ++k;
  }
  return b;
}
```

(We do not ask you to prove that the code equivalent, but you are welcome to think about it).

Write the corresponding RISC-V assembly code. What is the difference with the previous code?

c) Have a quick look on Godbolt (`https://godbolt.org/`, and select `RISC-V rv32gc gcc` under the list of compilers). What does the compiler do with the original code at `-O3`?

d) For the exam we will alternatively just ask to fill out the missing arguments in the following assembler code:



Exercise 3

a) Transform the following C code (where 0x is a prefix to indicate the numer is hexadecimal) into RISC-V assembly.

```
int a = 0xDEADBEEF;
```

b) Transform the following C code (where 0x is a prefix to indicate the numer is hexadecimal) into RISC-V assembly:

```
unsigned int a /*comes from context*/; signed int b;
if (a <= (unsigned)INT_MAX)
  b = (int) a;
else
  b = -1;
```

Compile the following code to RISC-V:

```
int a /*comes from context*/; signed int b;
if (a <= (int)INT_MAX)
  b = (int) a;
else
  b = -1;
```

How can it be optimized? In which of the two previous programs can the condition be replaced by `a < INT_MAX+1`?

c) Transform the following pseudocode into proper assembly

```
goto instruction such that PC = PC + 0xBADEAFFE;
goto instruction such that PC = PC + 2047;
```

d) For the exam we might instead ask to fill out the missing parts in the assembler program.

Original C program:

```
int a = 0xDEADB00F;
```

Corresponding machine code and assembler program:

```
0: deadb7b7          lui a5,0xdeadb
4: ___78793          addi a5,a5,0x7b7
```

No submission is needed. The exercise sheet will be discussed on November 10th, 2021, in-class and online
`https://uni-freiburg.zoom.us/j/65775356475?pwd=dmUvei8ybDN4RFlmT1JUZnRtYlBGZz09`

Category	Instruction	Example	Meaning	Comments
Arithmetic	Add	add x5, x6, x7	x5 = x6 + x7	Three register operands
	Subtract	sub x5, x6, x7	x5 = x6 - x7	Three register operands
	Add immediate	addi x5, x6, 20	x5 = x6 + 20	Used to add constants
	Set if less than	slt x5, x6, x7	x5 = 1 if x5 < x6, else 0	Three register operands
	Set if less than, unsigned	sltu x5, x6, x7	x5 = 1 if x5 < x6, else 0	Three register operands
	Set if less than, immediate	slti x5, x6, x7	x5 = 1 if x5 < x6, else 0	Comparison with immediate
	Set if less than immediate, uns.	sltiu x5, x6, x7	x5 = 1 if x5 < x6, else 0	Comparison with immediate
	Multiply	mul x5, x6, x7	x5 = x6 × x7	Lower 64 bits of 128-bit product
	Multiply high	mulh x5, x6, x7	x5 = (x6 × x7) >> 64	Upper 64 bits of 128-bit signed product
	Multiply high, unsigned	mulhu x5, x6, x7	x5 = (x6 × x7) >> 64	Upper 64 bits of 128-bit unsigned product
	Multiply high, signed-unsigned	mulhsu x5, x6, x7	x5 = (x6 × x7) >> 64	Upper 64 bits of 128-bit signed-unsigned product
	Divide	div x5, x6, x7	x5 = x6 / x7	Divide signed 64-bit numbers
	Divide unsigned	divu x5, x6, x7	x5 = x6 / x7	Divide unsigned 64-bit numbers
	Remainder	rem x5, x6, x7	x5 = x6 % x7	Remainder of signed 64-bit division
	Remainder unsigned	remu x5, x6, x7	x5 = x6 % x7	Remainder of unsigned 64-bit division
Data transfer	Load doubleword	ld x5, 40(x6)	x5 = Memory[x6 + 40]	Doubleword from memory to register
	Store doubleword	sd x5, 40(x6)	Memory[x6 + 40] = x5	Doubleword from register to memory
	Load word	lw x5, 40(x6)	x5 = Memory[x6 + 40]	Word from memory to register
	Load word, unsigned	lwu x5, 40(x6)	x5 = Memory[x6 + 40]	Unsigned word from memory to register
	Store word	sw x5, 40(x6)	Memory[x6 + 40] = x5	Word from register to memory
	Load halfword	lh x5, 40(x6)	x5 = Memory[x6 + 40]	Halfword from memory to register
	Load halfword, unsigned	lhu x5, 40(x6)	x5 = Memory[x6 + 40]	Unsigned halfword from memory to register
	Store halfword	sh x5, 40(x6)	Memory[x6 + 40] = x5	Halfword from register to memory
	Load byte	lb x5, 40(x6)	x5 = Memory[x6 + 40]	Byte from memory to register
	Load byte, unsigned	lbu x5, 40(x6)	x5 = Memory[x6 + 40]	Byte halfword from memory to register
	Store byte	sb x5, 40(x6)	Memory[x6 + 40] = x5	Byte from register to memory
	Load reserved	lr.d x5, (x6)	x5 = Memory[x6]	Load; 1st half of atomic swap
	Store conditional	sc.d x7, x5, (x6)	Memory[x6] = x5; x7 = 0/1	Store; 2nd half of atomic swap
	Load upper immediate	lui x5, 0x12345	x5 = 0x12345000	Loads 20-bit constant shifted left 12 bits
	Add upper immediate to PC	auipc x5, 0x12345	x5 = PC + 0x12345000	Used for PC-relative data addressing
Logical	And	and x5, x6, x7	x5 = x6 & x7	Three reg. operands; bit-by-bit AND
	Inclusive or	or x5, x6, x8	x5 = x6 | x8	Three reg. operands; bit-by-bit OR
	Exclusive or	xor x5, x6, x9	x5 = x6 ^ x9	Three reg. operands; bit-by-bit XOR
	And immediate	andi x5, x6, 20	x5 = x6 & 20	Bit-by-bit AND reg. with constant
	Inclusive or immediate	ori x5, x6, 20	x5 = x6 | 20	Bit-by-bit OR reg. with constant
	Exclusive or immediate	xori x5, x6, 20	x5 = x6 ^ 20	Bit-by-bit XOR reg. with constant
Shift	Shift left logical	sll x5, x6, x7	x5 = x6 << x7	Shift left by register
	Shift right logical	srl x5, x6, x7	x5 = x6 >> x7	Shift right by register
	Shift right arithmetic	sra x5, x6, x7	x5 = x6 >> x7	Arithmetic shift right by register
	Shift left logical immediate	slli x5, x6, 3	x5 = x6 << 3	Shift left by immediate
	Shift right logical immediate	srli x5, x6, 3	x5 = x6 >> 3	Shift right by immediate
	Shift right arithmetic immediate	srai x5, x6, 3	x5 = x6 >> 3	Arithmetic shift right by immediate
Conditional branch	Branch if equal	beq x5, x6, 100	if (x5 == x6) go to PC+100	PC-relative branch if registers equal
	Branch if not equal	bne x5, x6, 100	if (x5 != x6) go to PC+100	PC-relative branch if registers not equal
	Branch if less than	blt x5, x6, 100	if (x5 < x6) go to PC+100	PC-relative branch if registers less
	Branch if greater or equal	bge x5, x6, 100	if (x5 >= x6) go to PC+100	PC-relative branch if registers greater or equal
	Branch if less, unsigned	bltu x5, x6, 100	if (x5 < x6) go to PC+100	PC-relative branch if registers less
	Branch if greatr/eq, unsigned	bgeu x5, x6, 100	if (x5 >= x6) go to PC+100	PC-relative branch if registers greater or equal
Uncondit-ional branch	Jump and link	jal x1, 100	x1 = PC+4; go to PC+100	PC-relative procedure call
	Jump and link register	jalr x1, 100(x5)	x1 = PC+4; go to x5+100	Procedure return; indirect call

Figure 1: RISC-V instructions (taken from "Computer Organization and Design" by David A. Patterson, John L. Hennessy)