Given is the following assembly program:

```
0x00 ldi %r2, 1
0x04 ldi %r1, 1
0x08 mul %r2, %r2, %r0
0x0C sub %r0, %r0, %r1
0x10 cmp %r3, %r0, %r1
0x14 jgt %r3, -16
```

The first column of a line denotes the address of an instruction in memory, the second column the kind of instruction, and everything else that follows on a line are the operands.

# Exercise 4

Use the instruction set architecture in Section 1 to translate the assembly to machine instructions. The machine instructions must be written in binary format. Use 'X' to denote unused bits. A register maps to its corresponding binary number. For example, the assembly instruction `add %r7, %r8, %r9` translates to `00001001110100001001XXXXXXXXXXXX`.

## Submission

Hand in a plain text file with **ONLY** one machine instruction per line. Don't put the address before the instruction.

# Exercise 5

Assume that register `%r0` contains 2 before the program is executed. What would be the value in register `%r2` once the program reaches the instruction at address `0x18`, i.e. the branch at address `0x14` is not taken. What would be the value in `%r2`, if `%r0` would contain 3, 4, 9?

## Submission

Hand in a plain text file with the corresponding values per line, i.e. the value for 2 on the first line, the value for 3 on the second, etc.

# Exercise 6

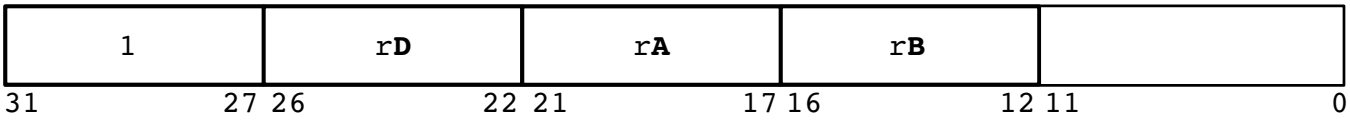
Assume that register `%r0` contains 1 before the program is executed. Given the microarchitecture in Section 2 and the control unit's state machine in Section 3, write down the values of the control unit's output signals for each executed instruction in the following order: imm, alu, regw, branch.

## Submission

Hand in a plain text file with the values for each executed instruction per line.

# 1 Instruction Set Architecture

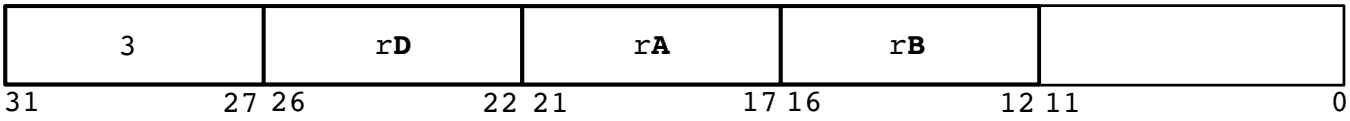
## add - Add Instruction




$r\mathbf{D} \leftarrow r\mathbf{A} + r\mathbf{B}$

The sum (r**A**) + (r**B**) is placed into r**D**. No other registers are altered.

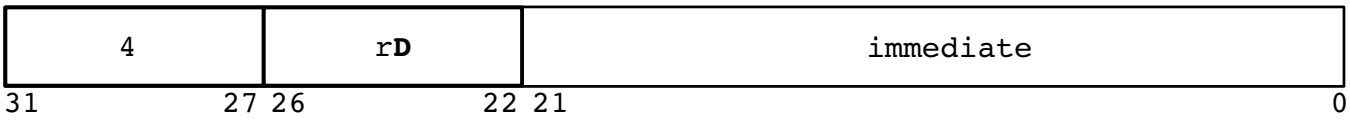
## cmp - Compare Instruction




$if\ r\mathbf{A} < r\mathbf{B}\ then\ r\mathbf{D}[0] \leftarrow 1\ else\ r\mathbf{D}[0] \leftarrow 0$
$if\ r\mathbf{A} > r\mathbf{B}\ then\ r\mathbf{D}[1] \leftarrow 1\ else\ r\mathbf{D}[1] \leftarrow 0$
$if\ r\mathbf{A} = r\mathbf{B}\ then\ r\mathbf{D}[2] \leftarrow 1\ else\ r\mathbf{D}[2] \leftarrow 0$
$r\mathbf{D}[3-31] \leftarrow 0$

The contents of r**A** are compared with the contents of r**B**, treating the operands as signed integers. The result of the comparison is placed into the three lowest order bits, and all other bits are cleared. No other registers are altered.

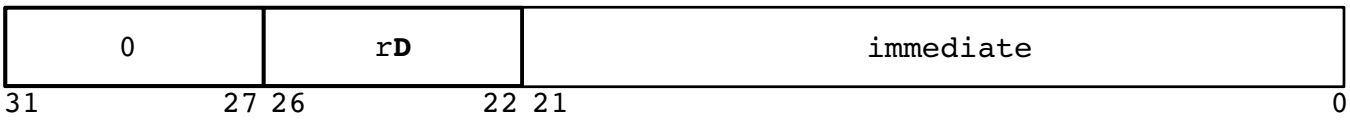
## jgt - Jump-Greater-Than Instruction




$if\ r\mathbf{D}[1] = 1\ then\ PC \leftarrow PC + immediate$

If r**D**[1] is set, PC is set to (PC) + (*immediate*). The *immediate* is treated as a signed integer. No other registers are altered.

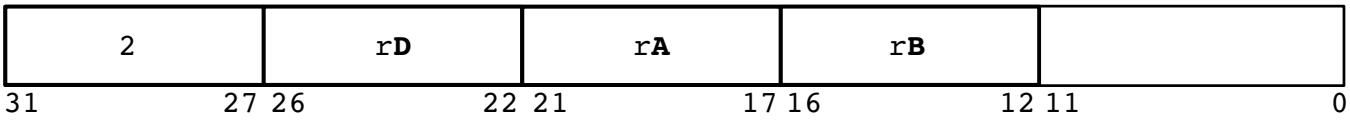
## ldi - Load Immediate Instruction




$r\mathbf{D} \leftarrow (0|immediate)$

The immediate is placed into r**D**[0-21]. The higher order bits of r**D** are cleared. No other registers are altered.

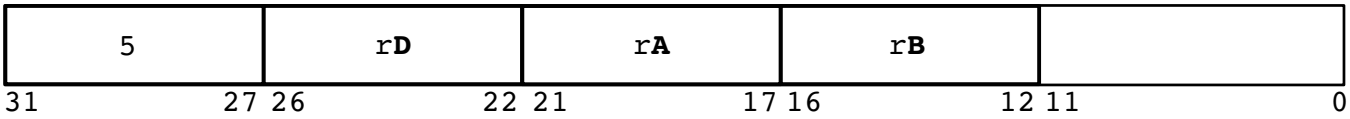
## mul - Multiplication Instruction




$r\mathbf{D} \leftarrow r\mathbf{A}[0-15] * r\mathbf{B}[0-15]$

The 32-bit operands are the contents of the low-order 16 bits of r**A** and r**B**. The low-order 32 bits of the 64-bit product r**A** * r**B** are placed into r**D**. The low-order 32-bits of the product are independent of whether the operands are regarded as signed or unsigned 32-bit integers. No other registers are altered.

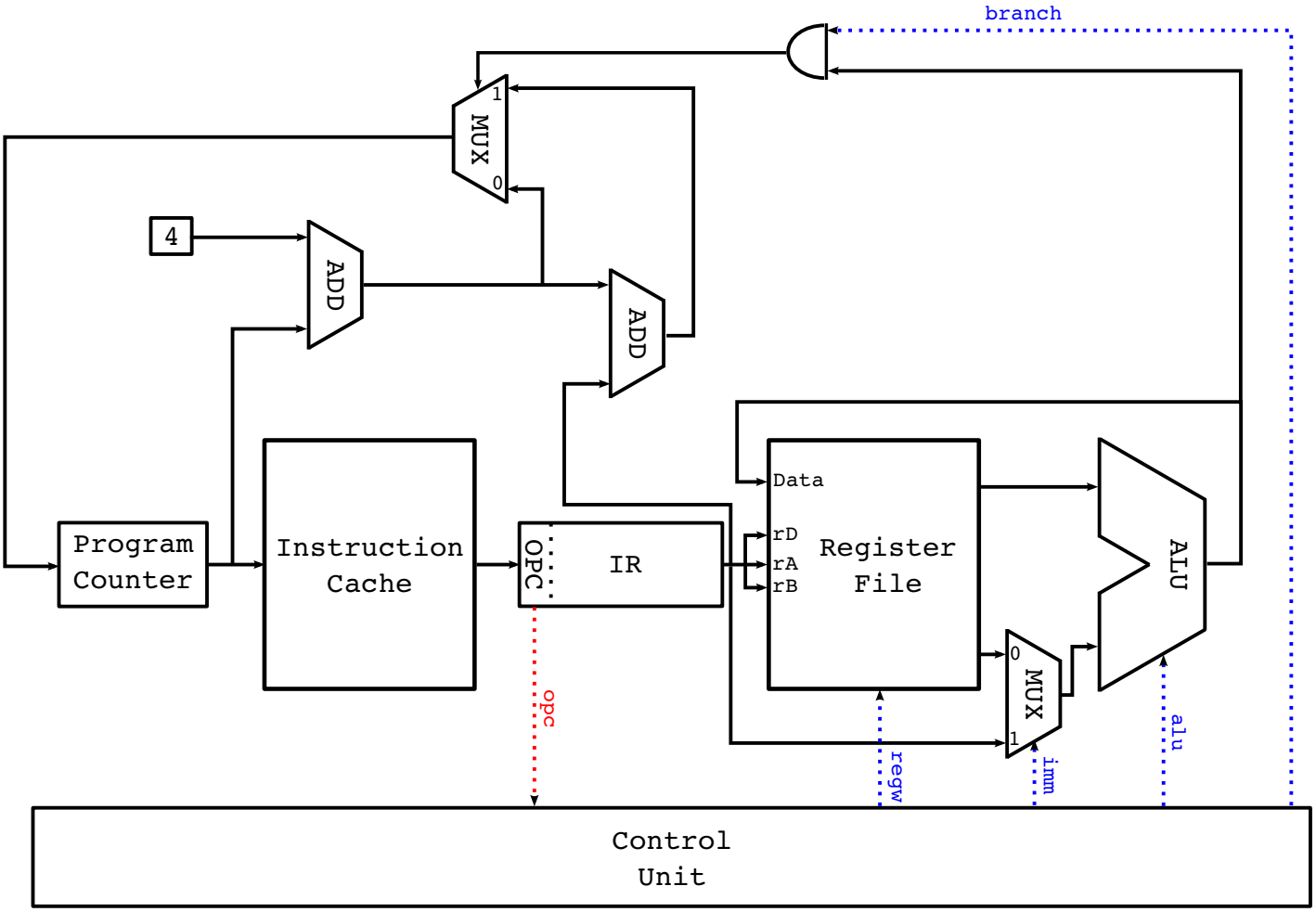
## sub - Subtraction Instruction




$r\mathbf{D} \leftarrow (r\mathbf{A}) - (r\mathbf{B})$

The difference (r**A**) - (r**B**) is placed into r**D**. No other registers are altered.

## 2 Microarchitecture




Given a program counter, i.e. a memory address, the cpu loads an instruction from the instruction cache into the instruction register. The control unit receives the opcode as input from the instruction register, and depending on the kind of instruction sets its output signals (regw, imm, alu, and branch) accordingly. See Section 3 for the control unit's state machine.

## 3 State Machine