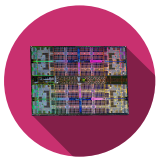


All Exercises CAr

Exercises Computer Architecture

1 | Chip & Die Fabrication

1.1 Fabrication

We wish to produce a chip that has a die size of 15mm x 20mm. The dies will be fabricated on 30cm diameter silicon wafers. We purchase ingots for 20'000CHF, each of which can be sliced into 100 wafers. We use a process which results in an average of 0.12 defects per cm^2

a) What is the fabrication yield?
b) How many dies can fit on a wafer?
c) How many good dies do we get per wafer?
d) What is the cost per good die?

fun/fabrication-01

1.2 Fabrication

We wish to produce a chip with a die size of $150mm^2$. The dies will be fabricated on 20cm diameter silicon wafers with a thickness of 5mm. We purchase ingots for 30'000CHF, each has a length of 60cm. We use a process which results in an average of 0.2 defects per cm^2, the package yield is 80% and the package cost are 0.06CHF per die.

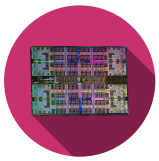
a) How many wafers per ingot?
b) What is the cost per wafer?
c) What is the cost per cm^2 of the wafer?
d) How many dies can fit on a wafer?
e) How many good dies on a wafer?
f) What does the final chip cost?

fun/fabrication-02

1.3 Fabrication

A process uses a 20cm diameter wafer and a $52.3mm^2$ die. The ingot costs 6000CHF and can be sliced into 30 wafers. The process results in 0.03 defects per mm^2.

a) How much does a wafer cost?
b) What is the Yield?
c) How many dies fit on a wafer?
d) What is the cost per die?



fun/fabrication-03

2 | Moore's Law & Denard scaling

2.1 Dennard Scaling

If the number of transistors per unit area (i.e., on the same chip size) doubles every 24 months:

a) By what factor does the length of the side of a transistor, assuming it is square, decrease every year?

b) Starting with a length of 18.4nm (2014), what would be the size of a transistor in 2025? How does this compare to the value of $6.75\mu m$?

fun/dennardscaling-01

2.2 Dynamic power consumption of a CMOS circuit is:

- ☐ directly proportional to the frequency
- ☐ inversely proportional to the frequency
- ☐ directly proportional to the square of the supply voltage

fun/dennardscaling-02

3 | Power Consumption

3.1 Cell phone battery life

A particular cell phone has an 8W-hr battery and operates at 0.707V. Suppose that, when it is in use, the cell phone operates at 2GHz. The total capacitance of the circuitry is 10 nF, and the activity factor is 0.05. When voice or data are active (10% of its time in use), it also broadcasts 3W of power out of its antenna. When the phone is not in use, the dynamic power drops to almost zero because the signal processing is turned off. But the phone also draws 100mA of quiescent current whether it is in use or not.
Determine the battery life of the phone:

a) if it is not being used

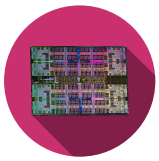
b) if it is being used continuously

fun/powerconsumption-01

1 | Processor Benchmark & Performance

1.1 Which of the following statements are correct?

- ☐ The wall clock time is the total elapsed time , including I/O, Operating system overhead etc.
- ☐ Multi-threading improves the throughput of a process
- ☐ The CPU time does not include the I/O time



- [ ] Multi-threading improves the execution time of a process

per/benchmark-01

1.2 What is the throughput?

- [ ] performance per Watt (the number of FLOPS per Watt)
- [ ] rate of processing work (n jobs/second)
- [ ] the time between start and completion of event/task/program (n seconds)
- [ ] the percentage of time a system is up and running

per/benchmark-02

1.3 What is the SPEC?

- [ ] is a benchmark suit developed to measure performance based on the latest Java application feature
- [ ] is a benchmark that evaluates the power and performance characteristics of volume server class computers
- [ ] is the wordwide standard for measuring graphics performance based on professional applications
- [ ] is a bechmark suit designed to provide performance measurements that can be used to compare computer-intensive workloads on different computer systems

per/benchmark-03

1.4 What is the goal of the EEMBC Benchmark?

- [ ] to evaluate performance of embedded microprocessors
- [ ] to evaluate integer computation performance
- [ ] to measure the energy efficiency of different computer systems
- [ ] to evaluate floating point performance

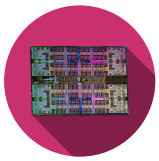
per/benchmark-04

1.5 Which of the following is an energy efficiency metric?

- [ ] flops
- [ ] MIPS
- [ ] Performance per watt
- [ ] Power consumption

per/benchmark-05

1.6 Both power consumption and performance per watt matters for an embedded system.



- ☐ True
- ☐ False

per/benchmark-06

1.7 Processor performance

A program consists of 5'000 floating point instructions and 25'000 integer instructions. Processor A has a clock rate of 2.0GHz. Floating point instructions take 7 cycles and integer instructions take 1 cycle.

a) How long does it take for this processor to run the program?
b) What is the average CPI for this processor for the given program?
c) Processor A runs program 2 consisting of 100'000 floating point instructions and 50'000 integer instructions. What is the average CPI for this program?
d) Processor B has an average CPI for program 2 of 3.5. It's clock rate is 1.8GHz. How much time does it take to execute the program?
e) Which processor is faster and by how much?

Processor _____ is _____ times faster than processor _____.

per/performance-01

1.8 Processor performance

Consider the following two machine designs with their respective CPI's for various instruction types. Computer A and Computer B have the same instruction set:

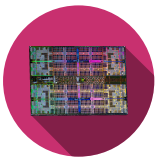
Instruction Type	CPI_A	CPI_B	Compiler 1 Mix
Data Manipulation	1.5	1.0	25%
Arithmetic	1.0	1.5	30%
Shifting	1.0	1.2	10%
Branching	4.0	2.0	25%
Multiply	20	12	10%

a) What is the average CPI for each of the computers using this program?
b) Computer A has a clock cycle time of 0.5ns. Computer B is running at 1.8GHz. Write a quantitative statement comparing the two computers.
c) What would the clock rate of the slower computer have to be to equal performaner of the faster computer?

per/performance-02

1.9 Processor performance

A CPU run on a base frequency of 2GHz. It executes a program with 5 million instructions with the given instruction mix. newline What is the execution time of the program?



Instruction	Frequency	CPI_{instr}
ALU	50%	3
Load	20%	5
Store	10%	4
Branch	20%	3

per/performance-03

1.10 Processor performance

A CPU is designed for an optimal performance on a given program with the following characteristics. 25% of all instructions are floating point instructions with an average CPI of 4.0, in addition the program contains 2% FPSQR instructions with an average CPI of 20. All other instructions have an average CPI or 1.33.
There are two design alternatives:
1. Decrease CPI of FPSQR instructions to 2.0
2. Decrease the average CPI of all floating point instructions to 2.5

Which choice is best?

per/performance-04

1.11 Processor performance

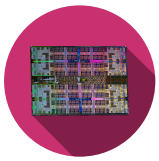
We want to buy a new computer. It will mostly run programs P_1 and P_2.
What weigth w_1 and w_2 need the programs to have so that:
a) CPU A the best buy?
b) CPU B the best buy?
c) CPU C the best buy?

Program	CPU_A	CPU_B	CPU_C
Program P_1 (sec)	1	10	100
Program P_2 (sec)	100	10	1

per/performance-05

1.12 Processor performance

Given the following performance of two programs on three CPU's, user the geometric mean to calculate which computer is the fastest:
a) CPU A is the fastest?
b) CPU B is the fastest?
c) CPU C is the fastest?



Program	CPU_A	CPU_B	CPU_C
P_1 (sec)	40	15	20
P_2 (sec)	40	1000	150

per/performance-06

1.13 Processor performance

Calculate the average CPI and the execution time for 5million instructions of the following instruction frequencies:

Instruction	Frequency	CPI_{instr}
ALU	40%	4
Load	30%	6
Store	5%	5
Branch	25%	4

The clock frequency of the CPU is 2 GHz

per/performance-07

1.14 What is the best metric for comparinc performance?

- ☐ arithmetic mean
- ☐ geometric mean
- ☐ median
- ☐ maximum performance
- ☐ harmonic mean

per/performance-08

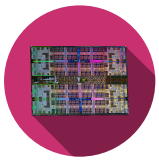
1.15 Processor performance

Calculate the execution time in ms, supposing to have CPU with the following instruction frequencies:

Instruction	Frequency	CPI_{instr}
ALU	45%	5
Load	25%	6
Store	10%	5
Branch	20%	3

For 2 Million instructions and a CPU frequency of 3GHz.

per/performance-09



1.16 Amdahl's Law

An improvement of the floating point execution unit resulted in 2x faster floating point instructions. In average 10% of all instructions are floating point instruction for this processor.

What will be the overall speedup?

per/amdahls-law-01

1.17 Amdahl's Law

We want an overall speedup of 2 and can accelerate the floating point instructions by 4 times.

What should be the fraction of floating point instructions?

per/amdahls-law-02

1.18 Amdahl's Law

A program consists of 2 different elements. Part A has a duration of 15 and part B a duration of 5 time units. There are two optimization variants:
1. optimization of the A part by two times
2. optimization of the B part by five times

Which optimization is more advantageous? What are the implications?

per/amdahls-law-03

1 Implementation

1.1 What is the main difference between a hard and a soft real-time system?

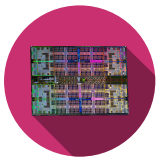
- ☐ in a hard-real-time system all deadlines must be met while in a soft-real time system occasionally some deadlines may be missed.
- ☐ in a soft-real-time system all deadlines must be met while in a hard-real time system occasionally some deadlines may be missed.

imp/implementation-01

1.2 What is an embedded system?

- ☐ Every computing system is an embedded system
- ☐ A computing system with a dedicated function, often with rea-time constraints, is an embedded system
- ☐ A general purpose computer with less than 1GB of RAM is an embedded system
- ☐ A system which has an ARM processor is an embedded system

imp/implementation-02



1.3 Faster execution time means less energy.

- ☐ True
- ☐ False

imp/implementation-03

1.4 Why are more and more SOC being developed insteead of CPU's?

- ☐ restricted availability of energy
- ☐ speed up often used functions
- ☐ both of the above

imp/implementation-04

1 | Instruction-Set Architecture

1.1 Simple C-Code to RISC-V Assembler

Compile the following C-Code into RISC-V assembler.

a)
```
a = b + c;
```

b)
```
a = b + c - d;
```

c)
```
a = b + 6;
```

d)
```
// int is a 32-bit signed word
int a = -372;
int b = a + 6;
```

e)
```
int a = 0xFEDC8765;
```

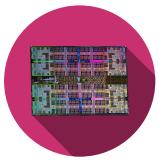
f)
```
int a = 0xFEDC8EAB;
```

isa/c-to-riscv-01

1.2 Algorithmic C-Code to RISC-V Assembler

Compile the following C-Code into RISC-V assembler.

a)
```
if (i == j){
  f = g + h;
}
f = f - i;
```



b)

```
if (i == j){
  f = g + h;
}
else {
  f = f - i;
}
```

c)

```
// add the numbers from 0 to 9
int sum = 0;
int i;

for (i=0; i!=10; i=i+1){
  sum = sum + i;
}
```

d)

```
// add the powers of 2 from 1 to 100
int sum = 0;
int i;

for (i=1;i<101; i=i*2){
  sum = sum + i;
}
```

e)

```
int array[5];
array[0] = array[0] * 2;
array[1] = array[1] * 2;
```

f)

```
int array[1000];
int i;

for (i=0; i<100; i=i+1){
  array[i] = array[i] * 8;
}
```

g)

```
char str[80] = "CAT";
int len = 0;

// compute length of string
while (str[len]) len++;
```

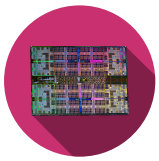
isa/c-to-riscv-02

1.3 Machine code to RISC-V Assembler

Decode the following Machine Code into RISC-V assembler.

a) **0x41FE 83B3**
b) **0xFDA4 8393**

isa/machinecode-to-riscv-01



1.4 Logic operations on registers

Execute the Assember code and specify the contents of the destination register **rd** if the source registers **rs** contain the following data:

```
s1 = 0x46A1 F1B7
s2 = 0xFFFF 0000
```

a)
```
and s3, s1, s2
```

b)
```
or s4, s1, s2
```

c)
```
xor s5, s1, s2
```

isa/riscv-execution-01

1.5 Logic operations on values

Execute the Assember code and specify the contents of the destination register **rd**, if the source registers **rs** contain the following data:

```
t3 = 0x3A75 0D6F
```

a)
```
and s5, t3, -1484
```

b)
```
or s6, t3, -1484
```

c)
```
xor s7, t3, -1484
```

isa/riscv-execution-02

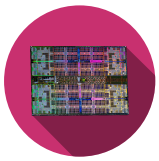
1.6 RISC-V multiplication

Execute the Assember code and specify the contents of the destination register **rd**, if the source registers **rs** contain the following data:

```
s1 = 0x4000 0000
s2 = 0x8000 0000
```

```
mulh s4, s1, s2
mul  s3, s1, s2
```

isa/riscv-execution-03



1.7 Division and modulo

Execute the Assember code and specify the contents of the destination register **rd**, if the source registers **rs** contain the following data:

```
s1 = 0x0000 0011
s2 = 0x0000 0003
```

```
div s3, s1, s2
rem s4, s1, s2
```

isa/riscv-execution-04

1.8 R-Type to Machine code

Encode the following RISC-V assembler into Machine Code.

a)
```
add s2, s3, s4
```

b)
```
sub t0, t1, t2
```

c)
```
sll s7, t0, s1
```

d)
```
xor s8, s9, s10
```

e)
```
srai t1, t2, 29
```

isa/riscv-to-machinecode-01

1.9 I-Type to Machine code

Encode the following RISC-V assembler into Machine Code.

a)
```
addi s0, s1, 12
```

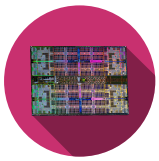
b)
```
addi s2, t1, -14
```

c)
```
lw t2, -6(s3)
```

d)
```
lh s1, 27(zero)
```

e)
```
lb s4, 0x1F(s4)
```

isa/riscv-to-machinecode-02



1.10 S-Type to Machine code

Encode the following RISC-V assembler into Machine Code.

a)
```
sw t2, -6(s3)
```

b)
```
sh s4, 23(t0)
```

c)
```
sb t5, 0x2D(zero)
```

isa/riscv-to-machinecode-03

1.11 Real-time system

What is the main difference between a “hard” real-time system and a “soft” real-time system?

- ☐ In a hard-real-time system all deadlines must be met while in a soft-real time system occasionally some deadlines may be missed.
- ☐ In a soft-real-time system all deadlines must be met while in a hard-real time system occasionally some deadlines may be missed.
- ☐ A game console must run under a “hard” real-time system, otherwise it’s impossible to run a game.
- ☐ A printer’s injection head must run under a “hard” real-time system, otherwise the printed page may be erroneous.
- ☐ An aircraft’s speed sensor must run under a “hard” real-time system, otherwise the autopilot may be disabled.
- ☐ A “soft” real-time system is faster than a “hard” real-time system.
- ☐ A “hard” real-time system is faster than a “soft” real-time system.
- ☐ Windows 10 is a hard real-time OS.
- ☐ Ubuntu Desktop is a hard real-time OS.
- ☐ RTLinux est un OS capable de travailler en temps réel “hard”.

isa/riscv-to-machinecode-04

1.12 U-Type to Machine code

Encode the following RISC-V assembler into Machine Code.

```
lui s5, 0x8CDEF
```

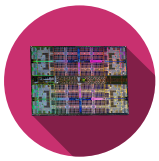
isa/riscv-to-machinecode-05

1.13 J-Type to Machine code

Encode the **first** jump instruction RISC-V assembler into Machine Code. The Program is as follows:

```

```



```
0x0000540C        jal ra,func1 # <--
0x00005410        add s1, s2, s3
...
0x001ABC04 func1: add s4, s5, s8
...
```

isa/riscv-to-machinecode-06

2 | Laboratory complement

To help you, feel free to use the RISC-V interpreter on https://course.hevs.io/car/riscv-interpreter/ as well as Ripes.

Be careful with variable types!

- **int** type is considered as a signed, 32 bits value.
- **unsigned int** type is considered as an unsigned, 32 bits value.
- If followed by a number (e.g.: **int16_t**), it means the value is on x bits (here 16). If preceded by a **u**, it is unsigned.

uint8_t is an unsigned byte, whereas **int8_t** is a signed byte.

2.1 Basic calculations

a)

```
int b = 1;
int c = 2;
a = b + c;

int b = -1;
int c = 2;
a = b + c;

int b = -12;
int c = 2023;
a = b + c;
```

b)

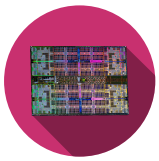
```
int b = 2;
int c = 3;
int e = -1;
int f = -78;
int g = 2023;
int h = -12;
a = b - c;
d = (e + f) - (g + h);
```

isa/lab-basic-calc

2.2 Memory access

```
uint16_t a = mem[3];
mem[4] = a;

int16_t a = mem[3];
mem[4] = a;
```



isa/lab-memory

2.3 Basic algorithms

1. Transmit the 8-bit memory value at address 0x0000′1000 serially, bit by bit, into the LSB of memory address 0x0000′1001. The remaining bits of memory address 0x0000′1001 must be **'0'**. Calculate the baud rate in $\frac{\text{Instructions}}{\text{Bit}}$ for the entire transmission.

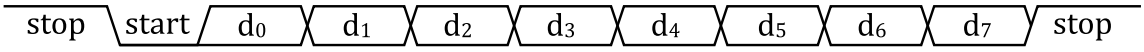


Figure 7: UART serial transmission

2. Multiply two 4-bit numbers together using one of the commands `bne` or `bge`. The algorithm works as follows: a multiplication is the same as adding the same number x times. For example: $2 * 9 = 9 + 9 = 18$.

isa/lab-basic-algos

2.4 Branching

2.4.1 If / else

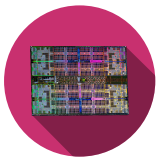
```
int a = 1, b = 2, c;

if(a == b) {
  c = 0;
} else if(a > b) {
  c = 1;
} else {
  c = 2;
}
```

2.4.2 Switch case

```
int a;

switch(mem[2]) {
  case 0:
    a = 17;
    break;
  case 3:
    a = 33;
    break;
  case 8:
  case 12:
    a = 10;
    break;
  default:
    a = 99;
}
```



2.4.3 While / Do While

```
// A
int a = 10;

do{a = a - 1;}
while(a != 0);

// B
int a = 10;

do{a = a - 1;}
while(a >= 0);

// C
unsigned int a = 10;

do{a = a - 1;}
while(a >= 0);
```

2.4.4 For

```
int a = 0, i;

for(i = 4; i > mem[0]; i = i - 1) {
  a = a + i;
}
```

isa/lab-branch

2.5 Functions

a)

```
int a = 1, b;
b = doubleIt(a);
b = doubleItOpti(a);

...

// Non-optimized version
// Let's assume a is saved in s0
int doubleIt(int myvar) {
  int a = myvar; // we WANT a in s0 !
  a = a * 2;
  return a;
}

// Optimized version
// Choose your registers freely
// Try having the less possible
//   instructions
int doubleItOpti(int myvar) {
  int a = myvar; //
  a = a * 2;
  return a;
}
```

b)

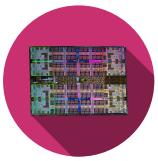
```
int a=1, b=2, c=3,
 d=4, e=5, f=6, g=7,
 h=8, i=9, j=10, res;
res = sum(a,b,c,d,e,f,
  g,h,i,j);

...

int sum(int v1, int v2,
  int v3, int v4, int v5,
  int v6, int v7, int v8,
  int v9, int v10){
  int c;
  c = v1 + v2 + v3 + v4 +
    v5 + v6 + v7 + v8 +
    + v9 + v10;
  return c;
}
```

isa/lab-fcts

2.6 Advanced Algorithmus



2.6.1 Modulo

The modulo % is an operation performed on two positive integers and is nothing other than the remainder of the division. For example, 5 divided by 3 gives 1 (you can pass 3 once in 5), **remains 2**. The modulo of a number by 0 is not defined.

The definition for signed numbers differs from language to language. We only deal with the unsigned version.

Modulo has many uses, allowing you to cap values, extract information, calculate an X and Y position from an X*Y value in an array of known size ...

- Give a code to perform this operation for any positive integer using the RV32IM set.
- How can the same thing be implemented in RV32I? Describe the concept(s).

The role of the compiler is to optimize the code as much as possible. If the operation detected is a modulo with a constant being a power of 2 (e.g. x % 2, y % 8 ...), a variant including no division is possible.

- Give this variant.

The notion of modulo for real numbers arrived with the evolution of computing power, and the result also diverges depending on the language. In C, a specific function from the std libraries is required, **fmod()**. In Python, this operation is native. In either case, they are more resource-intensive and require the handling of specific cases (NaN, infinity).

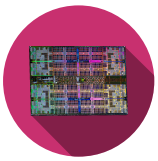
2.6.2 °F -> °C

We want to create a function capable of converting degrees Fahrenheit to Celsius. Since Fahrenheit values range from 32 to 1000, accuracy to the nearest degree is sufficient.
The formula is simple: $C = (F - 32) * \frac{5}{9}$
This would be handy if an FPU were available, but only the basic RV32I instruction set is supported. To get around this problem, you can use a few tricks whose algorithm is as follows:

A. Calculate $C = F - 32$.
B. Multiply by 5
C. Divide by 9
   - $\frac{1}{9}$ can be stored in a special binary representation

```
+-----+-----+-----+-----+-----+-----+------------+------------+
| b31 | b30 | b29 | b28 | b27 | ... |         b1 |         b0 |
+-----+-----+-----+-----+-----+-----+------------+------------+
| 2^0 | 2^-1| 2^-2| 2^-3| 2^-4| ... |      2^-30 |      2^-31 |
+-----+-----+-----+-----+-----+-----+------------+------------+
| 1   | 1/2 | 1/4 | 1/8 | 1/16| ... |1/1737418240|1/2147483648|
+-----+-----+-----+-----+-----+-----+------------+------------+
```

   in this case, it's $0000\_1110\_0011\_1000\_1110\_0011\_1000\_1110_2$.
   - The constant can be pre-calculated and is $\frac{2^n}{9} + 1$. The greater the n, the greater the precision. The number of bits defines the maximum size of n. The $+1$ is a rounding-off for lost precision.
   - Let's take $n = 16$. Our magic number is magic $= \frac{2^n}{9} + 1 = \frac{65536}{9} + 1 = 7282$.
   - Multiply the value by this magic number



D. Then divide by 2^n. In the case $n = 16 \rightarrow \frac{1}{65536}$.

To simplify the work, several assumptions are made:
- The magic number and the temperature in Fahrenheit are always positive.
- The size of the largest multiplication is :

$$\begin{aligned} \text{nbBits}_{\max_{\text{fahrenheit}}} + \text{nbBits}_{\text{mult5}} + \text{nbBits}_{\text{magicNumber}} = \\ 10(\text{max. 1000-32}) + 3 + (n - \text{nbBits}_{\text{div9}} + 1) = \\ 10 + 3 + (16 - 4 + 1) = \\ = 26 \text{ bits} \end{aligned} \tag{1}$$

  - It never exceeds 32 bits for $n < 23$.

Test and optimize the function:
- Write the corresponding code.
- Test with different Fahrenheit values.
- Test with several values for n (16, 18, 20). *Don't forget to recalculate the magic number.*
- Test the function with $n = [22, 23]$, for $°F = [100, 400, 1000]$.

isa/lab-adv-algos

1 | Architecture

1.1 Stack-Architecture

Evaluate the expression $\frac{a+bc}{a+dc-e}$ using a stack-based processor evaluation stack.

a) Write pseudo code of the calculation.
b) How many direct and indirect memory references are needed in case of an infinite stacksize?
c) How many direct and indirect memory references are needed in case of a stacksize of 2?

arc/stack-01

1.2 Stack-Architecture

Evaluate the expression $\frac{(a+b)^2}{\pi} * (a + b + c)$ using a stack-based processor evaluation stack.

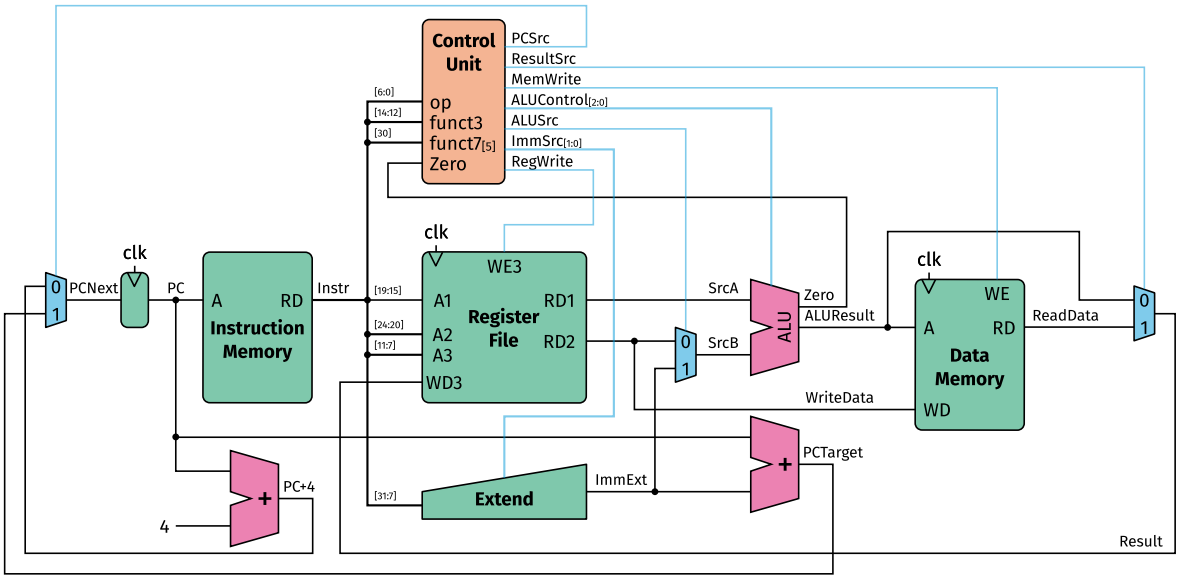
a) Write pseudo code of the calculation.
b) How many direct and indirect memory references are needed in case of a stacksize of 4?
c) How many direct and indirect memory references are needed in case of a stacksize of 3?
d) How many direct and indirect memory references are needed in case of a stacksize of 2?

arc/stack-02

2 | Single-Cycle RISC-V

2.1 Single-Cycle Processor Operation

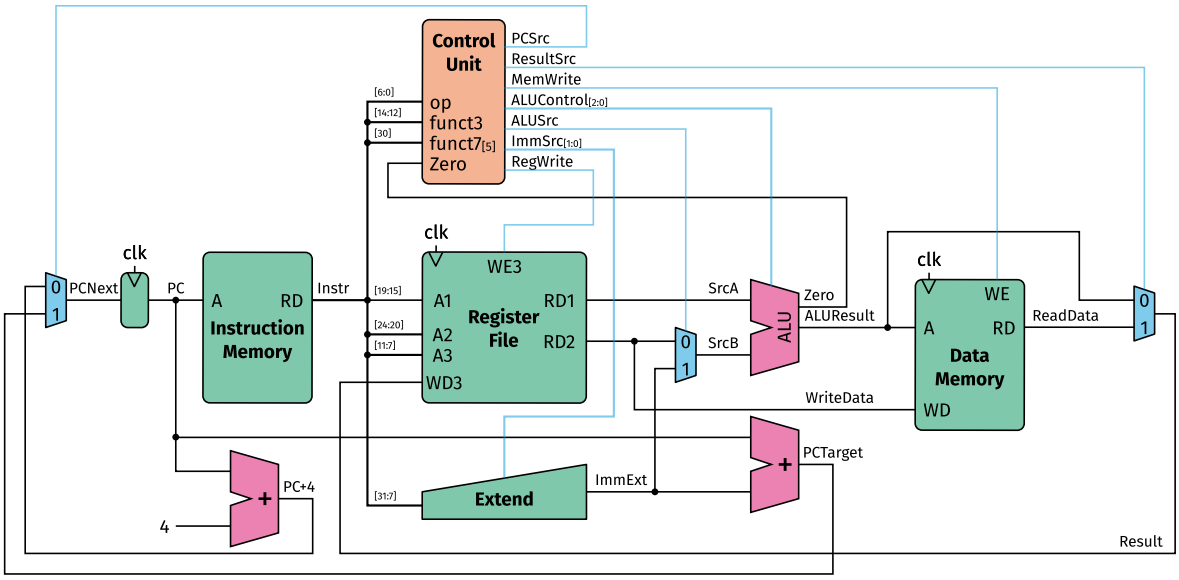
Determine the values of the control signals and the portions of the datapath that are used when executing an **and** instruction. Draw directly on the image the inner workings of the processor.



arc/scr-01

2.2 Extend Single Cycle with instruction `jal`

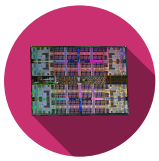
Show how to change the given RISC-V single-cycle processor to support the jump and link `jal` instruction. `jal` writes **`PC2+4`** to **`rd`** and changes the PC to the jump target address **`PC+imm`**.



arc/scr-02

2.3 Single Cycle Processor Performance

A single cycle processor build with a 7-nm CMOS manufacturing process has the following timing characteristics.



Element	Parameter	Delay(ps)
Register clk-to-Q	t_{pcq}	40
Register Setup	t_{setup}	50
Multiplexer	t_{mux}	30
AND-OR Gate	$t_{\text{AND\_OR}}$	20
ALU	t_{ALU}	120
Decoder (Control Unit)	t_{dec}	25
Extend Unit	t_{ext}	35
Memory Read	t_{mem}	200
Register File Read	t_{RFread}	100
Register File Setup	t_{RFSetup}	60

The program for the SPECINT2000 benchmark contains 100 billion instructions.
Calculate the execution time of the benchmark for this Single-Cycle Processor.

arc/scr-03

3 Multi-Cycle RISC-V

3.1 Multi Cycle Processor Performance

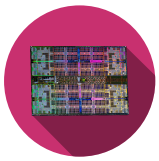
The program for the SPECINT2000 benchmark consists of approximately 25% loads, 10% stores, 11% branches, 2% jumps, and 52% R- or I-Type ALU Instructions.
Determine the average CPI for this benchmark for our developed Multi-Cycle Processor.

arc/mcr-01

3.2 Multi-Cycle Processor Performance

A multi cycle processor build with a 7-nm CMOS manufacturing process has the following timing characteristics.

Element	Parameter	Delay(ps)
Register clk-to-Q	t_{pc}	40
Register Setup	t_{setup}	50
Multiplexer	t_{mux}	30
AND-OR Gate	$t_{\text{AND\_OR}}$	20
ALU	t_{ALU}	120
Decoder (Control Unit)	t_{dec}	25
Extend Unit	t_{ext}	35
Memory Read	t_{mem}	200
Register File Read	t_{RFread}	100
Register File Setup	t_{RFSetup}	60



The program for the SPECINT2000 benchmark contains 100 billion instructions. Use the CPI_{avg} from the previous task.
Calculate the execution time of the benchmark for this multi-cycle Processor.

arc/mcr-02