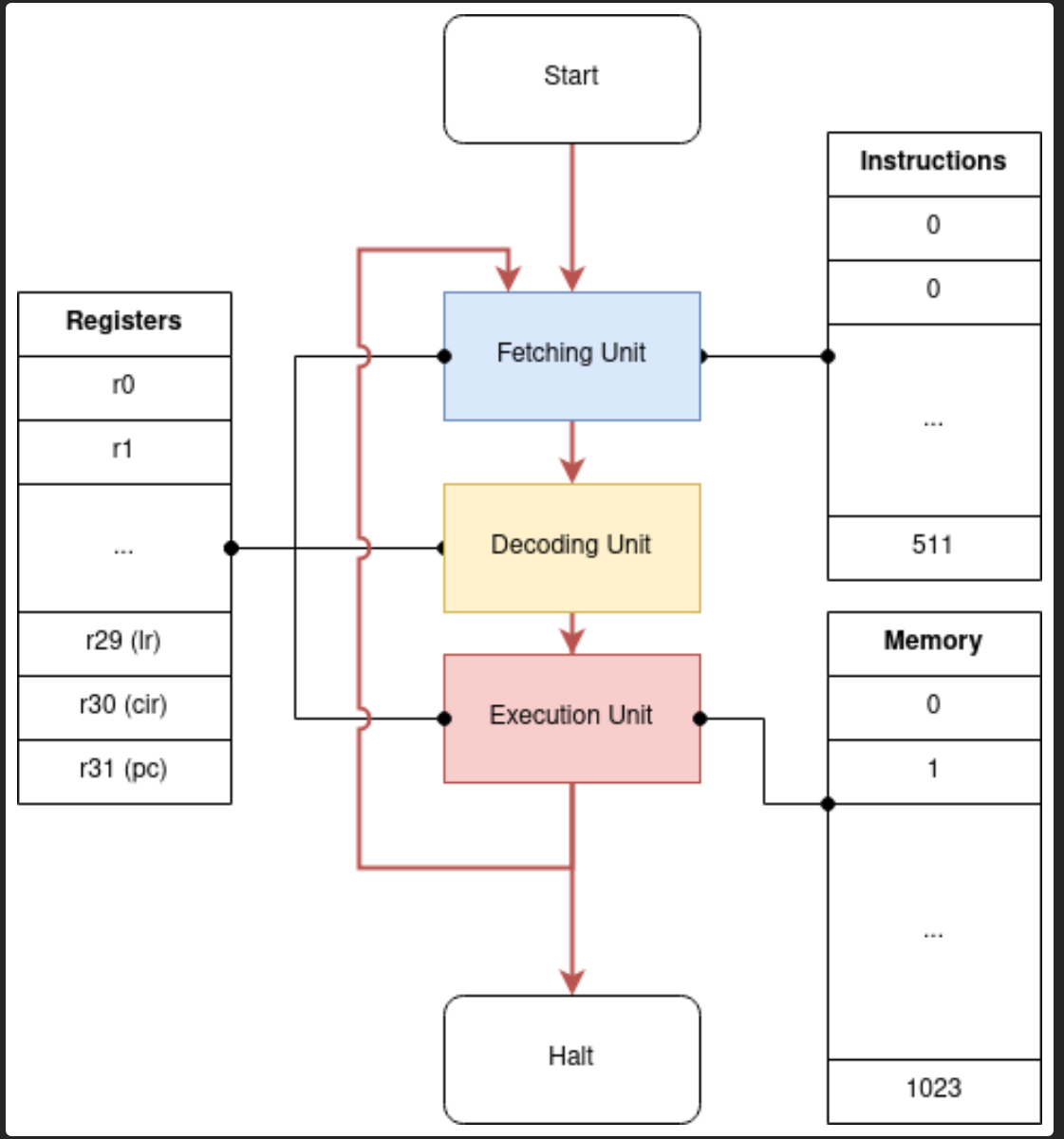
ACA Submission 1:

Scalar Processor Simulator

Tymoteusz Suszczynski

Processor Architecture

- The design follows a Harvard architecture and consists of 3 units for fetching, decoding and executing. The processor is scalar, so evaluating each of the units takes one cycle, meaning 3 cycles are spent per instruction.
- The fetching unit takes the instruction in INSTR[pc] and stores it in REG[30] (cir).
- The decoding unit takes the instruction in REG[30], decodes the instruction and passes the result to the execution unit.
- The execution unit carries out the instruction. It is able to manipulate the register file as well as memory.
- This cycle continues until the halt instruction is reached and execution stops.



Instruction Set Architecture

The following table describes the supported instructions and their function.

Opcode	Operands	Description
add	dest r1 r2	Add the contents of `r1` and `r2`, and store the result in `dest`.
addc	dest r1 #1	Add the contents of `r1` and an immediate `#1`, and store the result in `dest`.
mul	dest r1 r2	Multiplythe contents of `r1` and an immediate `#1`, and store the result in `dest`.
sub	dest r1 r2	Subtractthe contents of `r1` and an immediate `#1`, and store the result in `dest`.
ldm	dest src1 src2	Load from memory address `src1 + src2` , and store in `dest`.
ldc	dest #1	Store the `#1` immediate in `dest` register.
stm	src dest1 dest2	Store the contents of `src` in the memory address `dest1 + dest2`.
stmc	#src #dest1 #dest2	Like stm but all operands are immediates.
blt	dest cmp1 cmp2	Branch to `dest` if `cmp1 < cmp2`.
bnz	dest cmp1	Branch to `dest` if `cmp1 != 0`.
br	dest	Branch to `dest`.
cmp	dest a1 a2	Compares `a1` to `a2` and stores the result in `dest`. `0 => (a1 == a2)`. `-1 => (a1 < a2)`. `1 => (a1 > a2)`.
halt		Halt execution of the processor.

Running programs & usage

- The `assembler` and `proc` programs may be compiled using the Makefile:
  - `make proc`
  - `Make assembler`
- The submission contiains the assembly and binary code for 3 benchmarking programs:
  - `gcm.as`
  - `vector.as`
  - `bubble.as`
- Valid assembly files may be assembled to binary files with the `assembler` program included. To assemble `gcm`:
  - `./assembler gcm.as gcm.o`
- These programs may then be executed using the simulator. To execute the `gcm` program:
  - `./proc gcm.o`

Benchmarking: Vector addition

- This vector addition benchmark initialises two vectors in memory and adds them. The resulting vector is stored in memory, starting at location 10.

- The image shows the output of the simulator upon running the program. The vector from m10 to m14 is a sum of the vectors from m0 to m4 and m5 to m9.

- This benchmark completed in 139 cycles.

```
dziurawy-beret@pop-os:~/ACA2021/bin$ ./proc vector.o
m0: 1
m1: 2
m2: 3
m3: 4
m4: 5
m5: 1
m6: 2
m7: 3
m8: 4
m9: 5
m10: 2
m11: 4
m12: 6
m13: 8
m14: 10
Cycles: 139
```

Benchmarking: Bubble sort

- This bubble sort benchmark initialises an array {9, 5, 3, 8, 9, 32, 22, 43, 3, 60} and then sorts it using the bubble sort algorithm.

- The image shows the output of the simulator upon running the program. Only non-zero memory locations are shown. As we can see, the list is sorted.

- This benchmark completed in 1503 cycles.

```
dziurawy-beret@pop-os:~/ACA2021/bin$ ./proc bubble.o
m0: 3
m1: 3
m2: 5
m3: 8
m4: 9
m5: 9
m6: 22
m7: 32
m8: 43
m9: 60
Cycles: 1503
```

Benchmarking: Euclidean algorithm

- This benchmark calculated the greatest common divisor between two numbers.
- The output on the right shows m0 and m1 containing the two numbers upon which the algorithm is carried out, as well as m2 containing the result.

- This benchmark completed in 90 cycles.

```
dziurawy-beret@pop-os:~/ACA2021/bin$ ./proc gcd.o
m0: 52
m1: 18
m2: 2
Cycles: 90
```