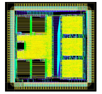
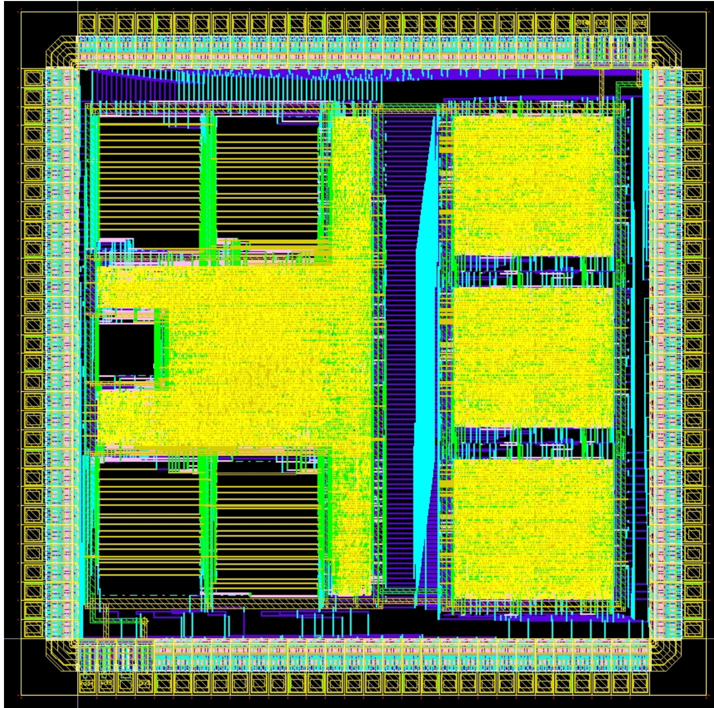
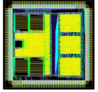


Digital Systems and Microprocessor Design (H7068)



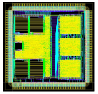
8.1. Intro to UoS educational processor

Daniel Roggen
d.roggen@sussex.ac.uk



UoS Educational Processor

- Developed in 2014 for "Digital Systems and Microprocessor Design" at University of Sussex by Daniel Roggen
- License: LGPL 2.1
- https://github.com/droggen/uos_educational_processor.git
- https://opencores.org/project,uos_processor



General processor

program counter

PC

D Q

Clk

+

1

A

Control

logic

Instruction

Dout

memory

a

we

din

Memory

dout

W

Register bank

ra

rb

rw

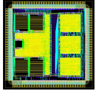
we

A B

op

ALU

- Instruction and data memory outside of processor
  - One reason: different semiconductor technologies
  - Higher densities with dedicated chips



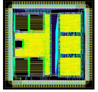
What's next: Educational Processor

- 8-bit, 4 register processor, Von Neumann architecture
- 3 clock cycle per instruction
- 16-bit instruction set (inspired by x86 ISA)
  - Direct, indirect, immediate, register addressing
- Customizable instructions

- External memory bus (for program/data)
- I/O interface (as in microcontrollers)

- Implemented in VHDL

- Synthesizable



CPU

ram_address
rst
ram_we
ram_datard
ext_in
ram_datawr
ext_out
clk

Fetch/exec seq
rst
clk
fetchh
fetchl
exec

memory write decode logic

d
en
rst
q
clk

instruction(15..8)
instruction(7..0)
instruction(15..0)

exec

1
c_o + c_i
1 0
s
jump

1 0
s

d
en
rst
q
clk
1
5
ip(4..0)

instruction(15..8)
jump decode logic

Register bank
d
rwr
rwren
rrd1
rrd2
rst
clk
q1
q2
8
2
reg1out
reg2out

instruction(9..8)
register write decode logic
instruction(9..8)
instruction(1..0)

data address generation

instruction(7..0)
instruction(12)

1 0
s
source

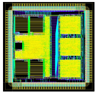
instruction(14..10)
instruction(15..10)
instruction(15..13)

A
B
op
f
q
4
8
aluqout

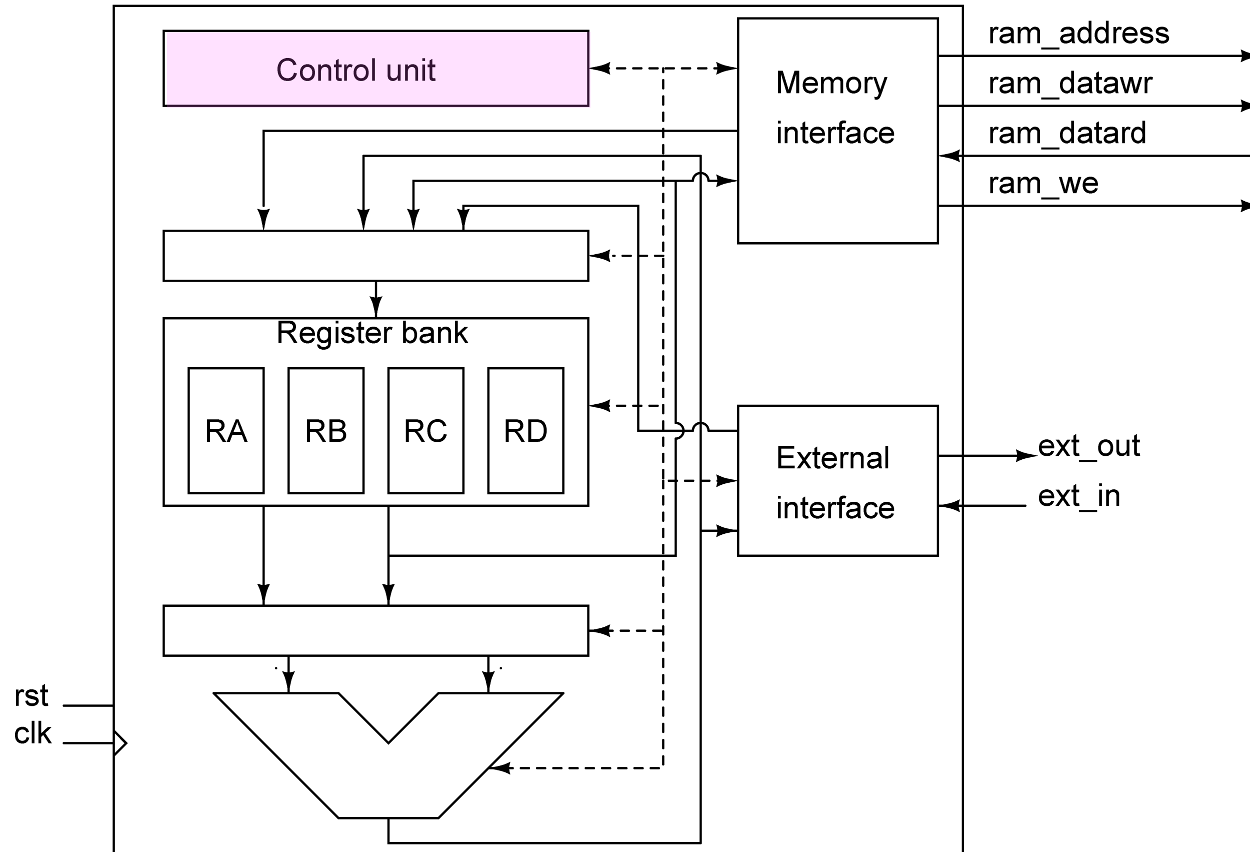
write data select
wrdata(7..0)
exec

ext write decode logic
d
en
rst
q
clk

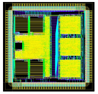
flag write decode logic
d
en
rst
q
clk
flags(3..0)



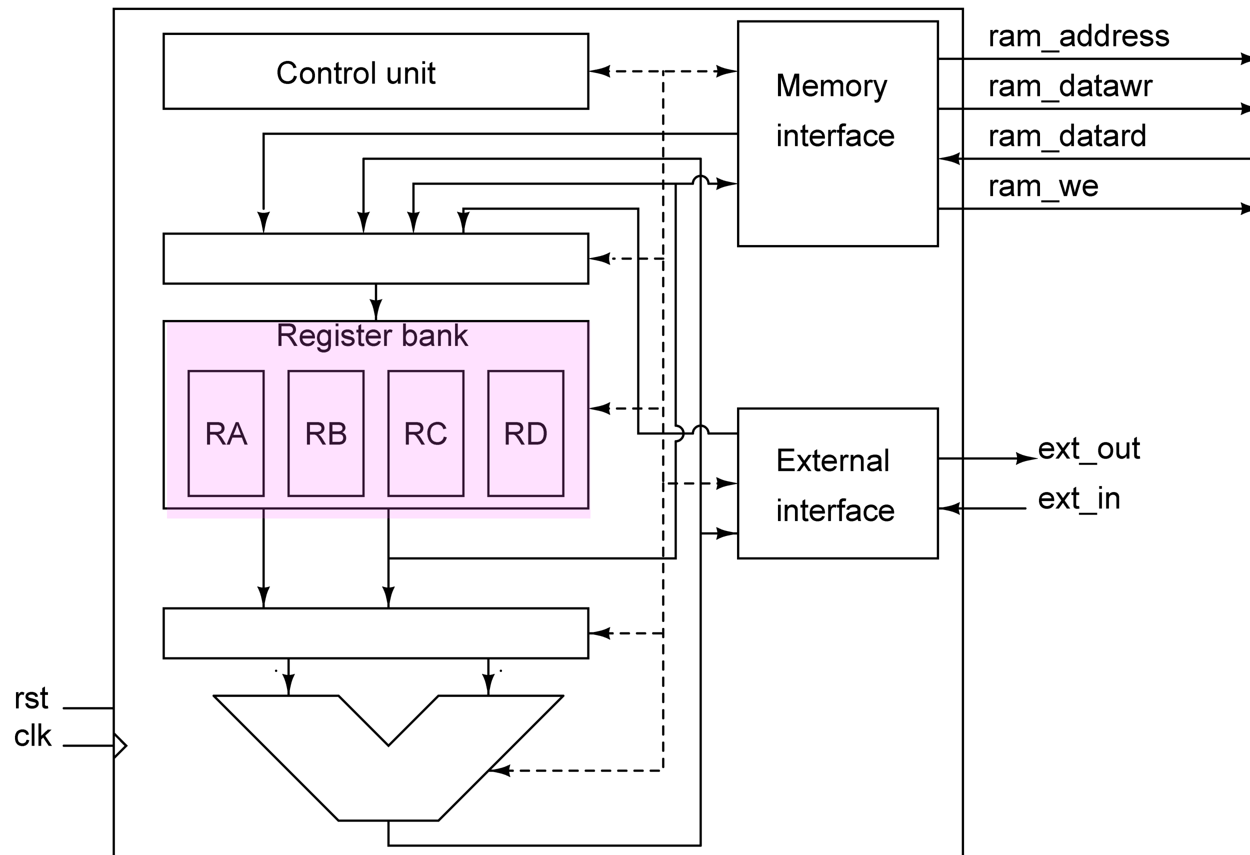
Educational processor overview



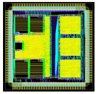
- Control unit: loads the instruction from memory and controls its execution
  - Memory interface: whether to read or write memory (we: write enable), where to read (address), the data to write (datawr) and gets the data from memory with datard
  - External interface: whether to write or read
  - The ALU operation
  - The input to the register bank
  - The input to the ALU



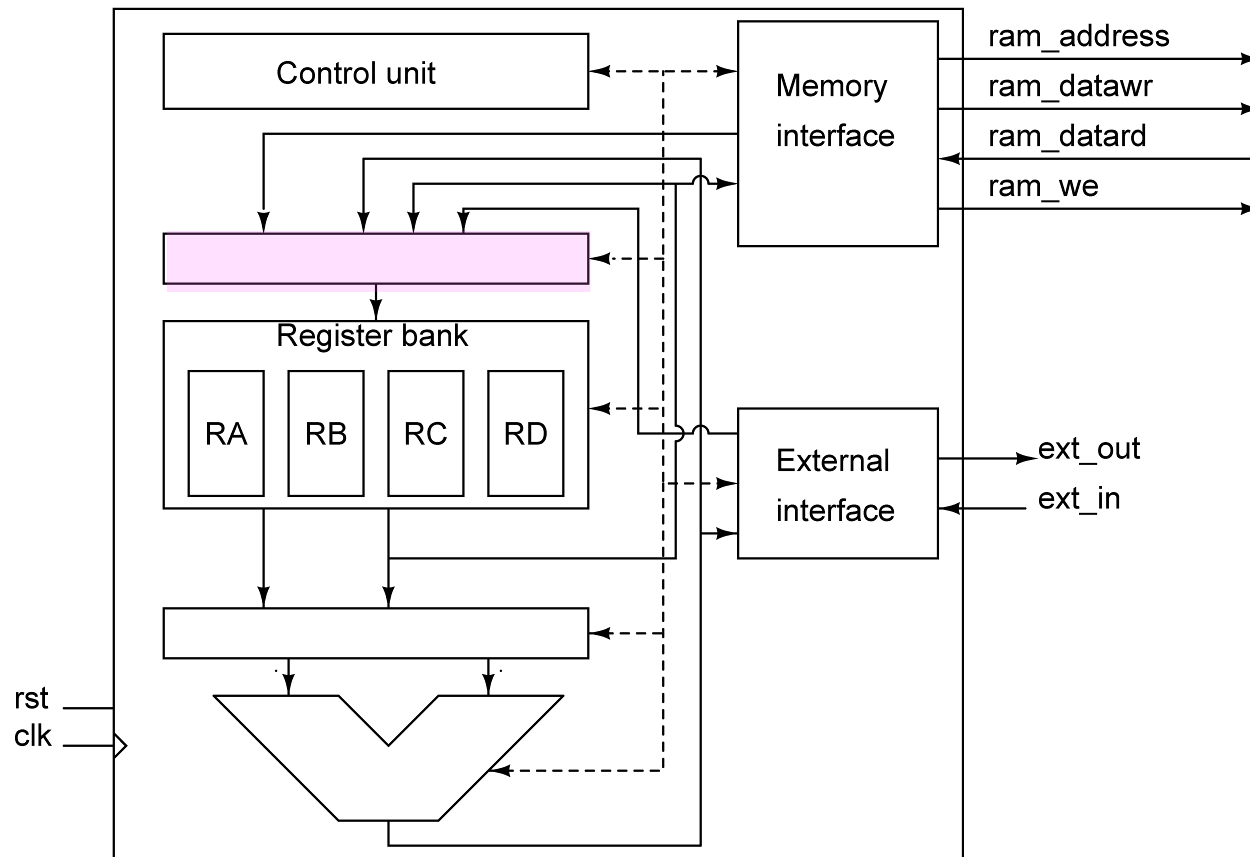
Educational processor overview



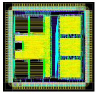
- Register bank: 4 8-bit registers RA,RB,RC,RD
  - Provides two outputs from any two registers (asynchronous output). Registers to read from are specified by control lines.
  - Allows to synchronously write to one register. Register to write to, and whether to write, is controlled by control lines.
  - Registers (to read or write) are identified by a 2-bit code: RA=00, RB=01, RC=10, RD=11



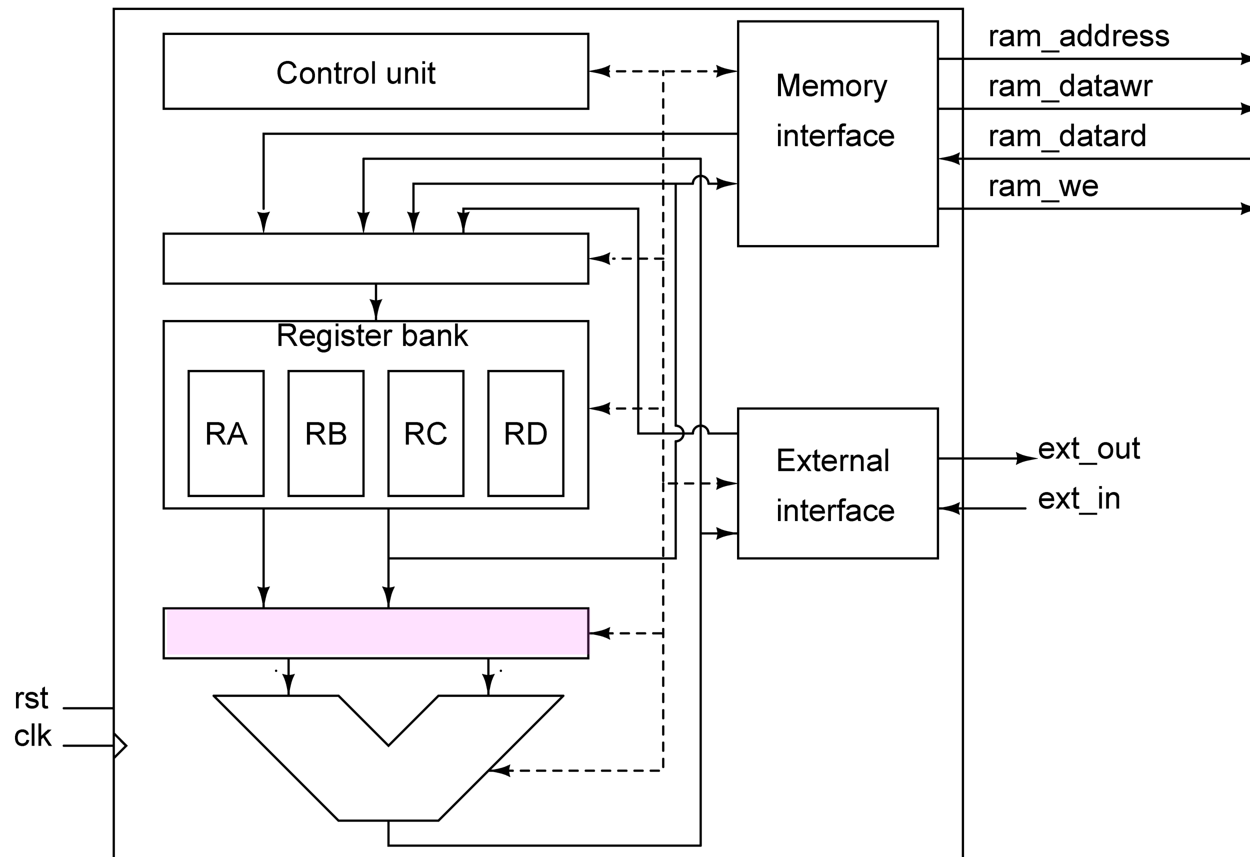
Educational processor overview



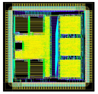
- Register bank input: select data written to the register bank (if a write occurs in the register bank)
- Data to register bank can come from: ALU output, memory interface, external interface, another register, or the instruction



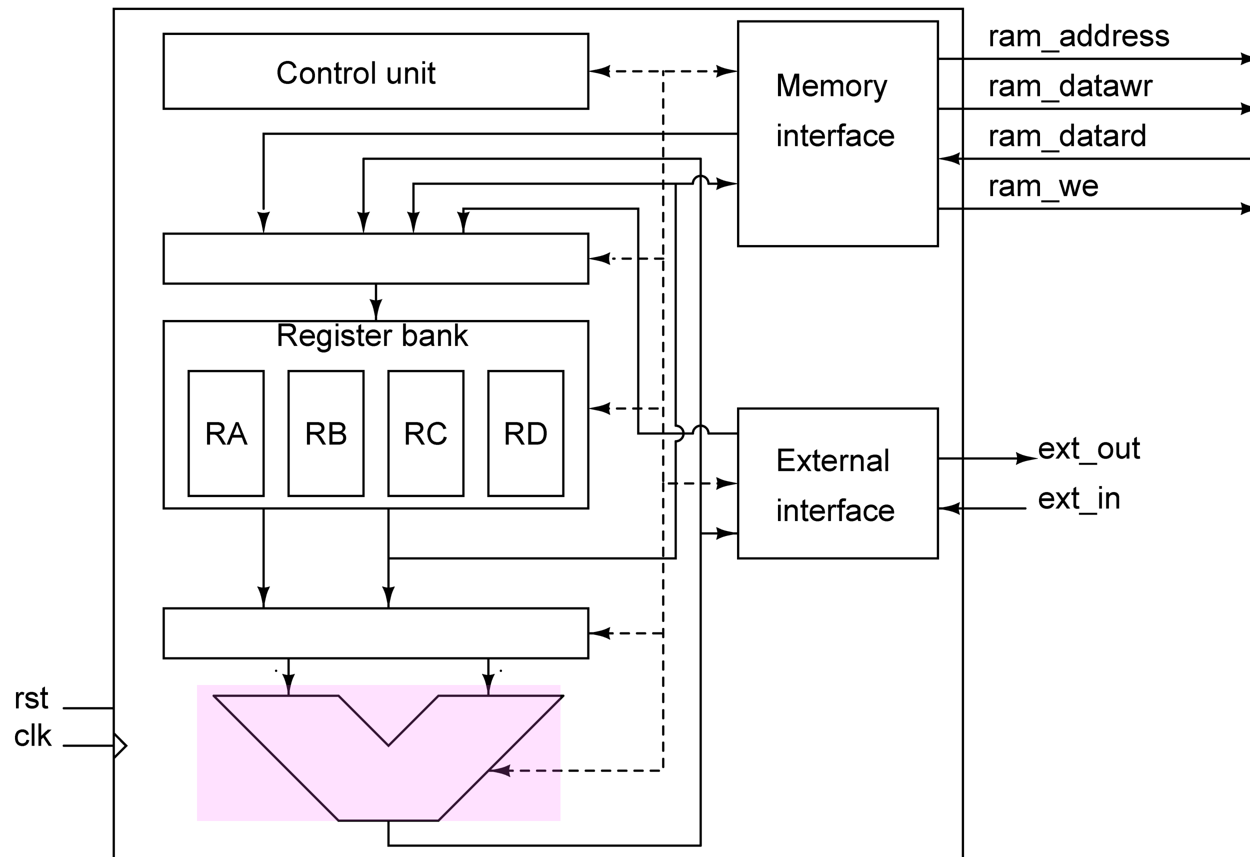
Educational processor overview



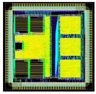
- ALU input: select data fed to ALU
- ALU input data can come from the register bank or from the instruction



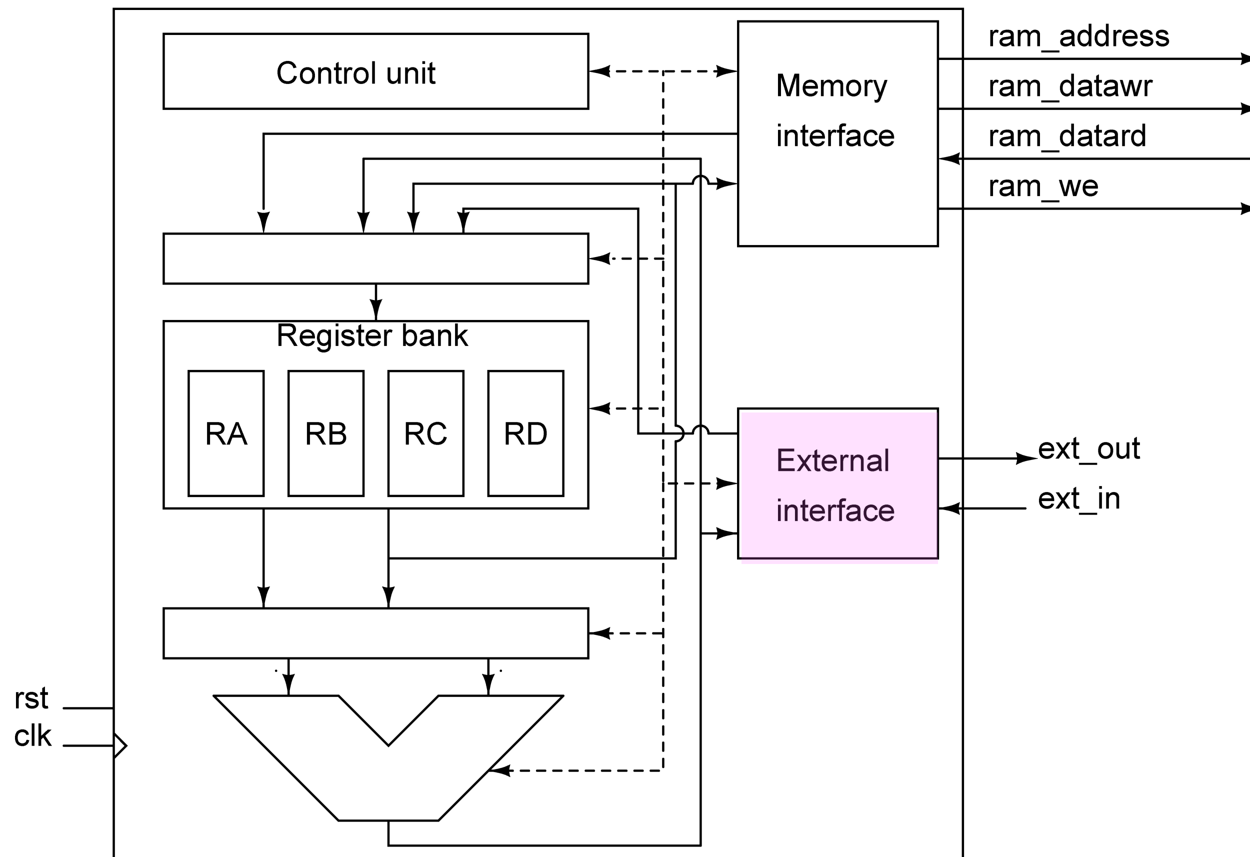
Educational processor overview



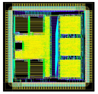
- The ALU takes two inputs and performs a logical or arithmetic operation defined according to the control line



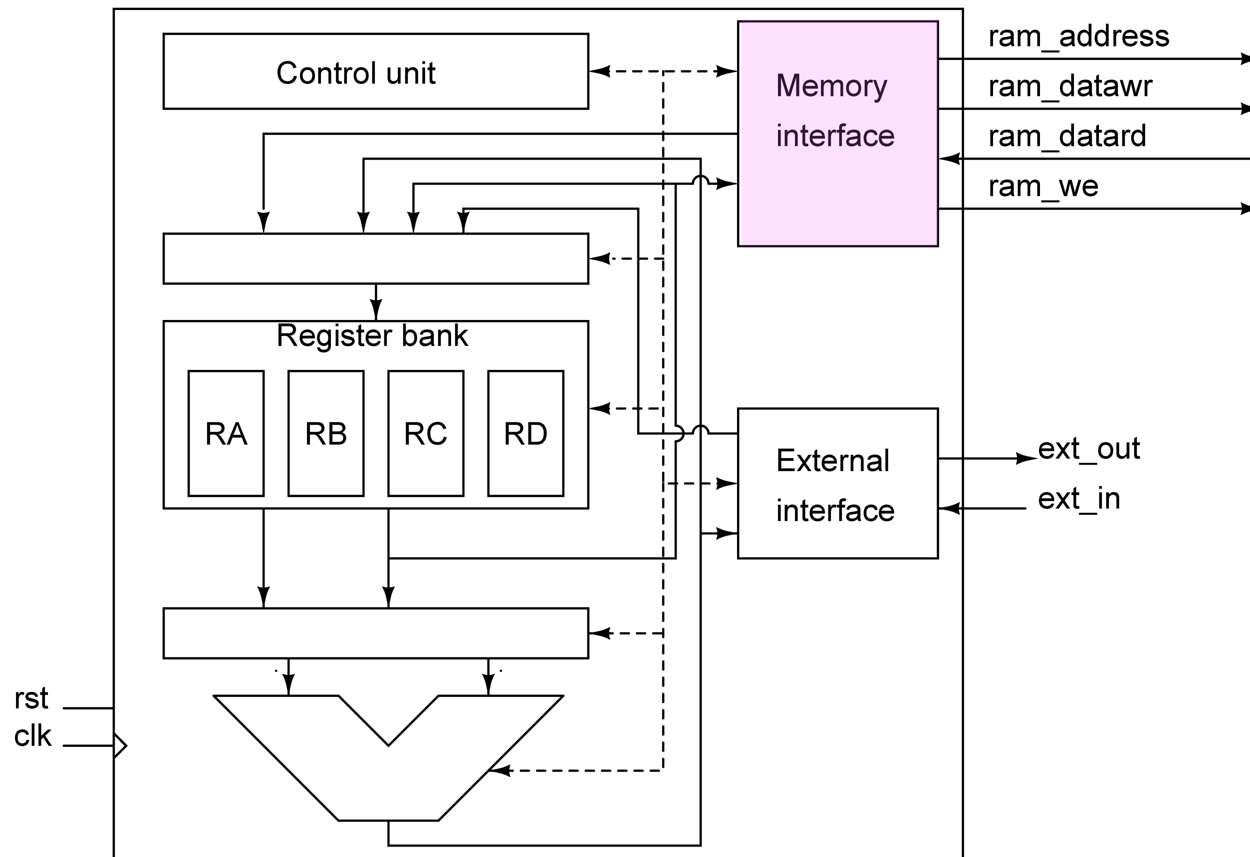
Educational processor overview



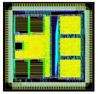
- The external interface allows to read or write data from a parallel interface on the processor
- This is commonly used in microcontrollers to connect peripherals (LEDs, buttons)
- During a write the interface stores the value to put on ext_out (D FF).
- No special operation during read, however more advanced external interfaces could perform signal conditioning (e.g. debouncing)



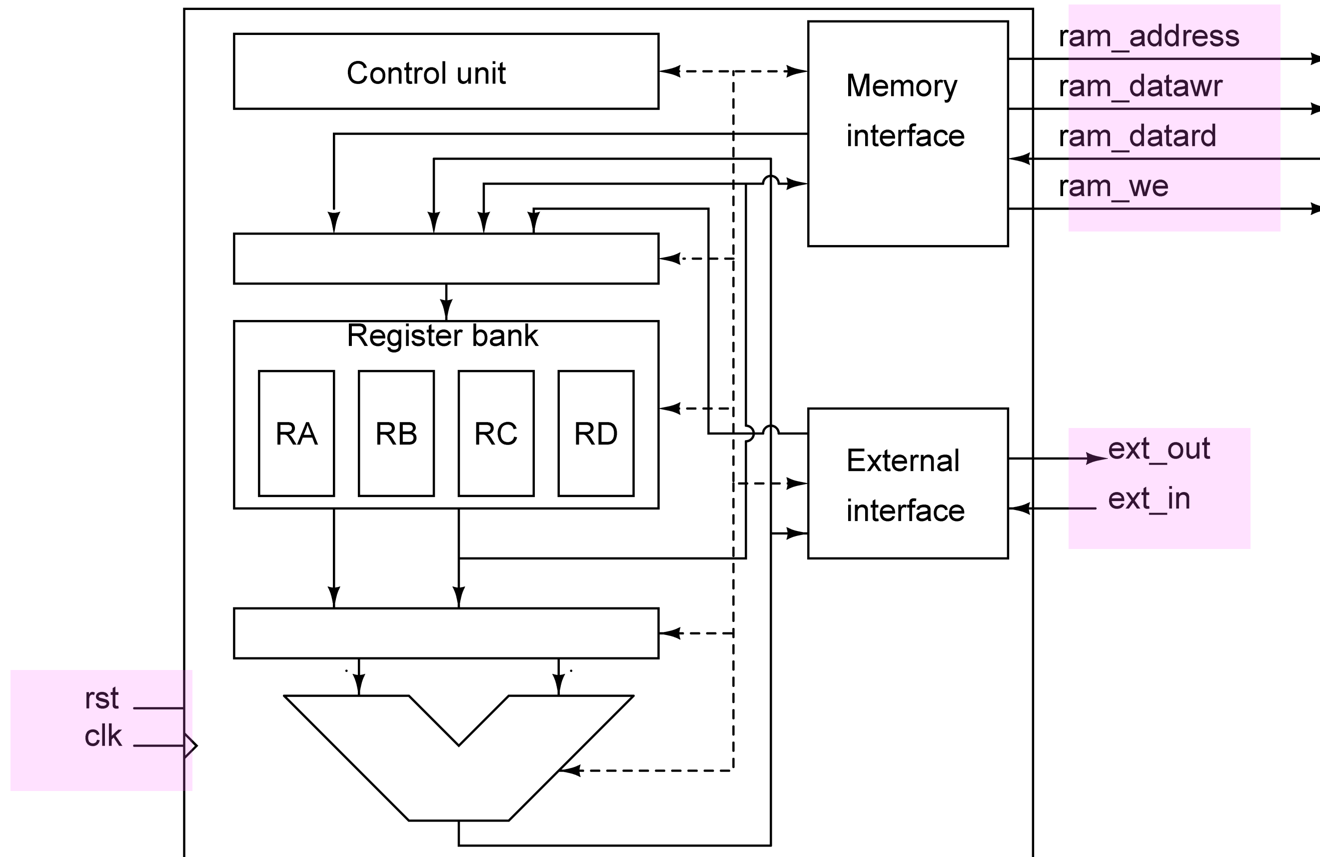
Educational processor overview



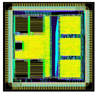
- The memory interface allows to connect to an external memory
- In this educational processor the memory interface is transparent (no special function).
- More advanced processors may have special interfaces to read from DRAM, SRAM, SD cards, etc.



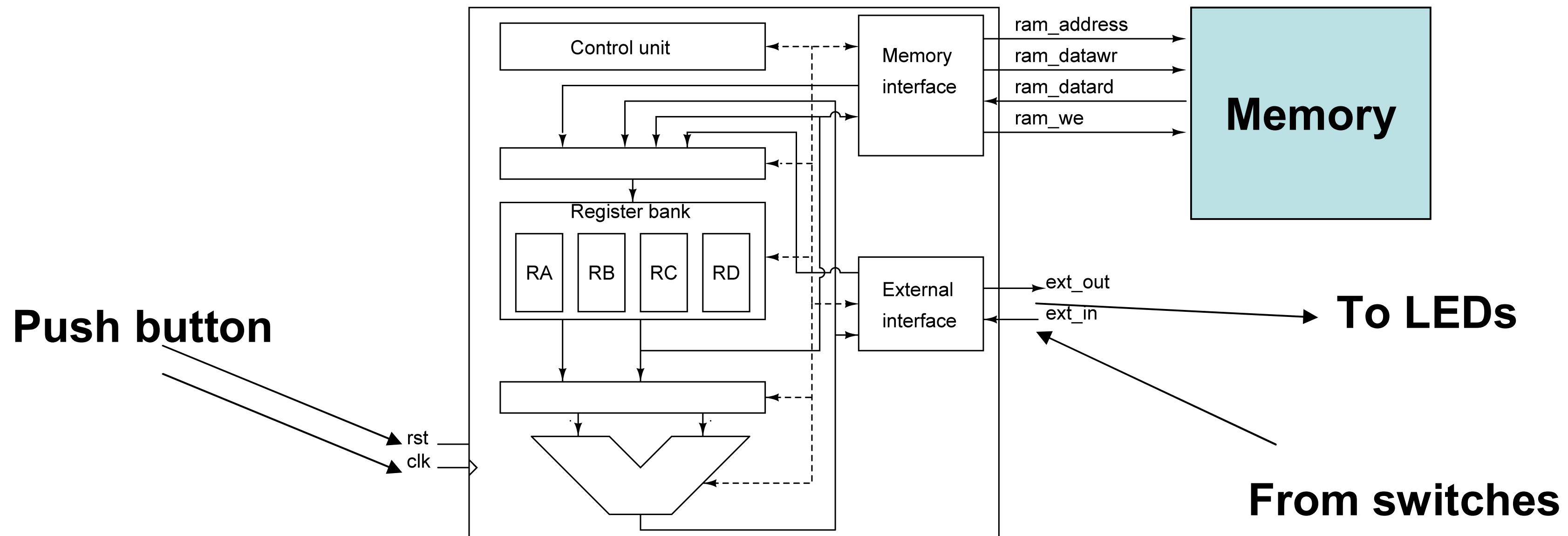
Educational processor overview



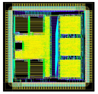
- Processor ports: memory interface, exernal interface, clock and reset
- Reset is synchronous! (occurs on the rising edge of clock)
- More advanced processors may have several memory and external interfaces, additional pins to generate "interrupts" (branch of the execution flow when a pin is toggled), etc.



Educational processor on FPGA

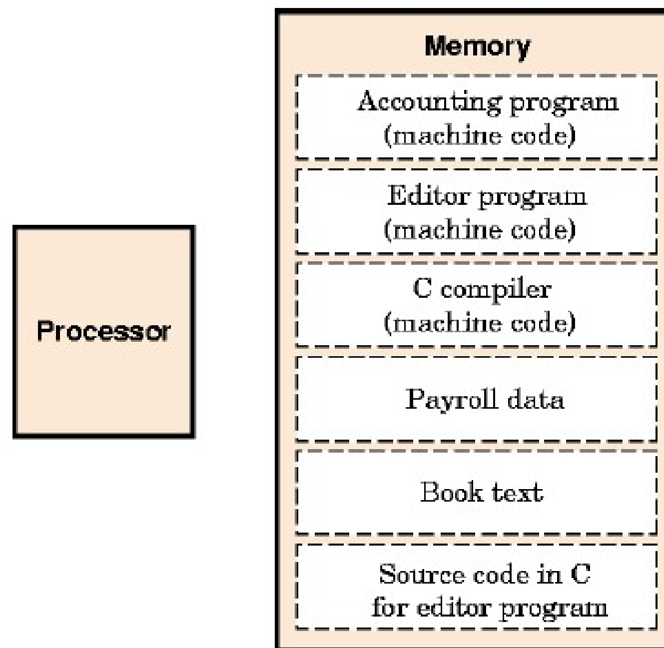


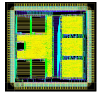
- The processor is synthesized on the FPGA as any other component with the **`entity port map`** syntax.
- A 32 bytes memory is synthesized alongsize the processor for program and data. It has 32 entries (5 bit address) of 8 bits
- Push buttons allow to generate single clocks to test the processor
- LED and switches connected to external interface



Stored program (von Neumann)

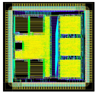
- Instructions represented as number in memory
- Programs are just like data
- However:
  - Program goes to the control unit
  - Data goes to the data path





Instruction

- An instruction defines the operation of the processor when it is executed
- An instruction is defined by it's bit-width and whether it is fixed-length or variable length
  - Fixed length lead to easier implementation but use more memory
  - Variable length can optimize the size for frequent instruction
- It comprises multiple fields: opcode (operation code), source, destination, etc.
- Different processor architectures have different instruction sets with their own encoding

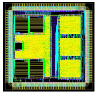


Instructions in the educational CPU

- All instructions are 16-bit wide (fixed size)

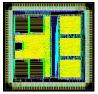
instr(15..13)	instruction(12..8)	instruction(7..0)
Opcode	depends on the instruction	src

- 3-bit opcode (operation code): indicates the type of operation
- The meaning of the remaining bits depends on the opcode!



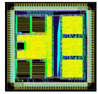
Program counter / Instruction pointer

- PC or IP: register in the processor control unit that indicates the memory location where the instruction is fetched
- PC starts at zero on reset
- As instructions are 16 bits, the first instruction is at memory location 00 and 01; the second instruction at memory location 02 and 03;...
- PC incremented continuously for usual instructions
- Except with "jump" instructions: the PC changed to fetch instruction from another location



Encoding v.s. "assembler" instruction

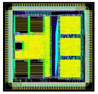
- All instructions are 16-bit data stored in memory
- The instructions can be specified by their binary code:
  - 1000001010101
- Or to simplify reading by their hex code:
  - 1055h
- To further simplify reading we use a human readable format:
  - mov ra,55h
- We refer to this format as an "assembler" instruction because a software (or human) would read the text "**`mov ra,55h`**" and "assemble" the various parts of the encoding to obtain 1055h



Opcodes

- Defines the "category" of the instruction
- 3-bit opcode: total of 8 "categories" of instructions
- Defined in order to help the decoding of the instruction.
- All instructions of the same opcode share the same encoding

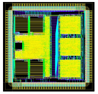
- Opcode 000: move instructions
- Opcode 001: ALU instructions
- Opcode 010: ALU instructions
- Opcode 011: ALU instructions
- Opcode 100: unused
- Opcode 100: ALU instructions
- Opcode 101: jump instructions
- Opcode 110: external interface instructions
- Opcode 111: unused



Move instructions (opcode 000)

- Moves data between registers, immediate and memory
- mov dst,src
  - moves the data specified by source into destination

Instructions	instruction(15..8)								Instruction(7..0)							
Move	Opcode			$\overline{R}/I$	dd#m	sd#m	dreg		src							
mov r, r	0	0	0	0	0	0	r	r	-	-	-	-	-	-	r	r
mov r, i	0	0	0	1	0	0	r	r	i	i	i	i	i	i	i	i
mov r, [r]	0	0	0	0	0	1	r	r	-	-	-	-	-	-	r	r
mov r, [i]	0	0	0	1	0	1	r	r	i	i	i	i	i	i	i	i
mov [r], r	0	0	0	0	1	0	r	r	-	-	-	-	-	-	r	r
mov [r], i	0	0	0	1	1	0	r	r	i	i	i	i	i	i	i	i

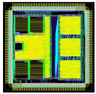


immediate/register

- the src field contains the "source" data for the instruction (sometimes unused)
- Source can be immediate or register depending on R'/I

Instructions	instruction(15..8)					Instruction(7..0)
Move	Opcode	$\overline{R}/I$	dd#m	sd#m	dreg	src

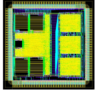
- R'/I=1: src is an immediate: the 8 LSBs in the instruction are used as the data
- R'/I=0: src is a register: the data comes from a register. The register is specified by the 2 least significant bits in src
  - RA: 00
  - RB: 01
  - RC: 10
  - RD: 11



direct/indirect

- Source: direct or memory mode specified by sd#m
- Direct mode: the value of a register or immediate is moved to dst
- Memory mode: the instruction fetch the data from the memory location specified by src (which can be immediate or register)
- Syntax: use brackets around src to indicate memory mode
  - mov ra,[55h]

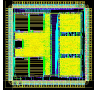
Instructions	instruction(15..8)					Instruction(7..0)
Move	Opcode	$\overline{R}/I$	dd#m	sd#m	dreg	src



destination

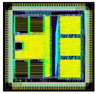
- Destination is always a register (direct) or a memory location (memory mode) specified by a register, depending on dd#m
- Direct mode: the value of source is moved to a register
- Memory mode: the instruction will fetch the data from a memory location.
- Syntax: use brackets around dst to indicate memory mode
  - mov [ra],55h

Instructions	instruction(15..8)					Instruction(7..0)
Move	Opcode	$\overline{R}/I$	dd#m	sd#m	dreg	src



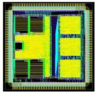
Move examples

- mov ra,rb:
  - src is direct, register: moves the content of reg b into reg a
  - dst is direct register
- mov ra,13h:
  - src is direct, immediate: moves 13h into reg a
  - dst is direct register
- mov ra,[rb]
  - src is memory, register: moves the data at the memory location b into a
  - dst is direct register
- mov ra,[13h]:
  - src is memory, immediate: moves the data at the memory location 13h into a
  - dst is direct register
- mov [ra],rb:
  - src is register:
  - dst is memory: moves the content of reg b into reg a



ALU (opcodes 001,010,011)

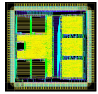
- Performs an arithmetic/logic operation on one or two operands
- instr dst, src
  - Performs a two operand operation on dst and src and puts the result in dst
- instr dst
  - Performs a single operand operation on dst and puts the result in dst
- dst is always a register
- src is a register or an immediate



ALU: two operands

- ALU op indicates which ALU operation
- src: source (immediate/register according to R'/I)
- dst: destination

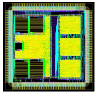
Instructions	instruction(15..8)								Instruction(7..0)							
ALU 2 op	opcode			$\overline{R}/I$	ALU op		dreg		src							
add r, r	0	0	1	0	0	0	r	r	-	-	-	-	-	-	r	r
add r, i	0	0	1	1	0	0	r	r	i	i	i	i	i	i	i	i
sub r, r	0	0	1	0	0	1	r	r	-	-	-	-	-	-	r	r
sub r, i	0	0	1	1	0	1	r	r	i	i	i	i	i	i	i	i
and r, r	0	0	1	0	1	0	r	r	-	-	-	-	-	-	r	r
and r, i	0	0	1	1	1	0	r	r	i	i	i	i	i	i	i	i
or r, r	0	0	1	0	1	1	r	r	-	-	-	-	-	-	r	r
or r, i	0	0	1	1	1	1	r	r	i	i	i	i	i	i	i	i
xor r, r	0	1	0	0	0	0	r	r	-	-	-	-	-	-	r	r
xor r, i	0	1	0	1	0	0	r	r	i	i	i	i	i	i	i	i



Examples

- add RA,RB
  - Stores RA+RB in RA
- sub RA,03h
  - stores RA-3 in RA.
- and RD,55h
  - stores the logical AND of RD and 55h in RD

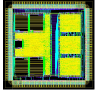
Always indicate numbers by a 2 digit with an h at the end for hex!
Avoids confusion between register RA and value Ah



ALU: comparison

- Comparison: cmp dst,src
- src: source (immediate/register according to R'/I)
- dst: destination
- Comparison is performed by subtracting src from dst!
- Result of the comparison is stored in flags: carry and zero
  - Zero=1 Carry=0: dst=src
  - Zero=0 Carry=0: dst>src
  - Zero=0 Carry=1: dst<src
- Result of comparison used by conditional jump

Test	opcode			$\overline{R}/I$	ALU op		dreg		immedite / reg							
cmp r, r	0	1	0	0	0	1	r	r	-	-	-	-	-	-	r	r
cmp r, i	0	1	0	1	0	1	r	r	i	i	i	i	i	i	i	i

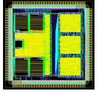


ALU: one operand

- Format: instr dst
- The operation is applied on dst and the result is in dst

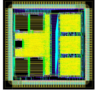
Instructions	instruction(15..8)								Instruction(7..0)							
ALU 1 op	opcode			ALU op			dreg									
not r	0	1	1	0	0	0	r	r	-	-	-	-	-	-	-	-
shr r	0	1	1	0	0	1	r	r	-	-	-	-	-	-	-	-
ror r	0	1	1	0	1	0	r	r	-	-	-	-	-	-	-	-
asr r	0	1	1	0	1	1	r	r	-	-	-	-	-	-	-	-
rol r	0	1	1	1	0	0	r	r	-	-	-	-	-	-	-	-

- Example:
  - not ra
  - asr rb



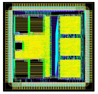
Jumps (opcode 101)

- Unconditional jumps: changes the value of PC to destination
  - jmp dst
- Conditional jumps: changes the value of PC if a condition is met. Condition is tested by checking the flags (carry, zero). Flags are set by a prior comparison
- JA: jump if above
  - Jumps if Zero=0 and Carry=0
- JB: jump if below
  - Jumps if Zero=0 and Carry=1
- JE: jump if equal
  - Jumps if Zero=1



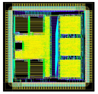
Compare / jump examples

		Carry	Zero
`mov`	`ra,0Ah`	0	0
`cmp`	`ra,09h`	0	0
`cmp`	`ra,0Ah`	0	1
`cmp`	`ra,0Bh`	1	0



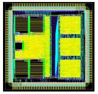
Compare / jump examples

```
                             Carry      Zero
mov  ra,0Ah                  0          0
cmp  ra,09h                  0          0
jb   dst1          -- not taken: RA not below 9h
je   dst2          -- not taken: RA not equal 9h
ja   dst3          -- taken: RA above 9h
```



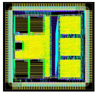
Compare / jump examples

```
                              Carry     Zero
mov  ra,0Ah                   0         0
cmp  ra,0Ah                   0         1
jb   dst1        -- not taken: RA not below Ah
je   dst2        -- taken: RA not equal Ah
ja   dst3        -- not taken: RA not above Ah
```



Compare / jump examples

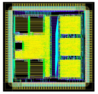
```
                            Carry      Zero
mov   ra,0Ah                0          0
cmp   ra,0Bh                1          0
jb    dst1       -- taken: RA below Bh
je    dst2       -- not taken: RA not equal Bh
ja    dst3       -- not taken: RA above Bh
```



External interface (opcode 110)

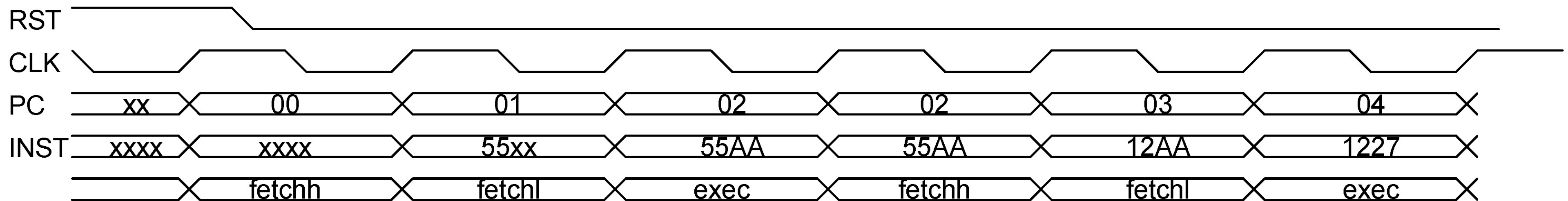
Instructions	instruction(15..8)								Instruction(7..0)							
IO	opcode			$\overline{R}/I$	IO type		dreg		src							
out r	1	1	0	0	0	0	-	-	-	-	-	-	-	-	r	r
out i	1	1	0	1	0	0	-	-	i	i	i	i	i	i	i	i
in r	1	1	0	-	0	1	r	r	-	-	-	-	-	-	-	-

- Out: write register or immediate to the external interface
- In: read data from the external interface into a register



Instruction fetch and execution

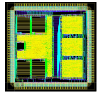
- Instructions are 16 bit but memory is 8 bit!
- Two cycles needed to fetch the instruction
- One cycle for execution



Memory dump:

Address	Content
0:	55
1:	AA
2:	12
3:	27

- Consequence: 3 clock cycles per instruction



Summary

- The architecture and features of the processor are clear at a high level
- The characteristics of the instruction set are understood:
  - Instruction execution time
  - Instruction encoding
  - Instruction set (move, alu, jump, external) and its characteristics (register/immediate, direct/memory)