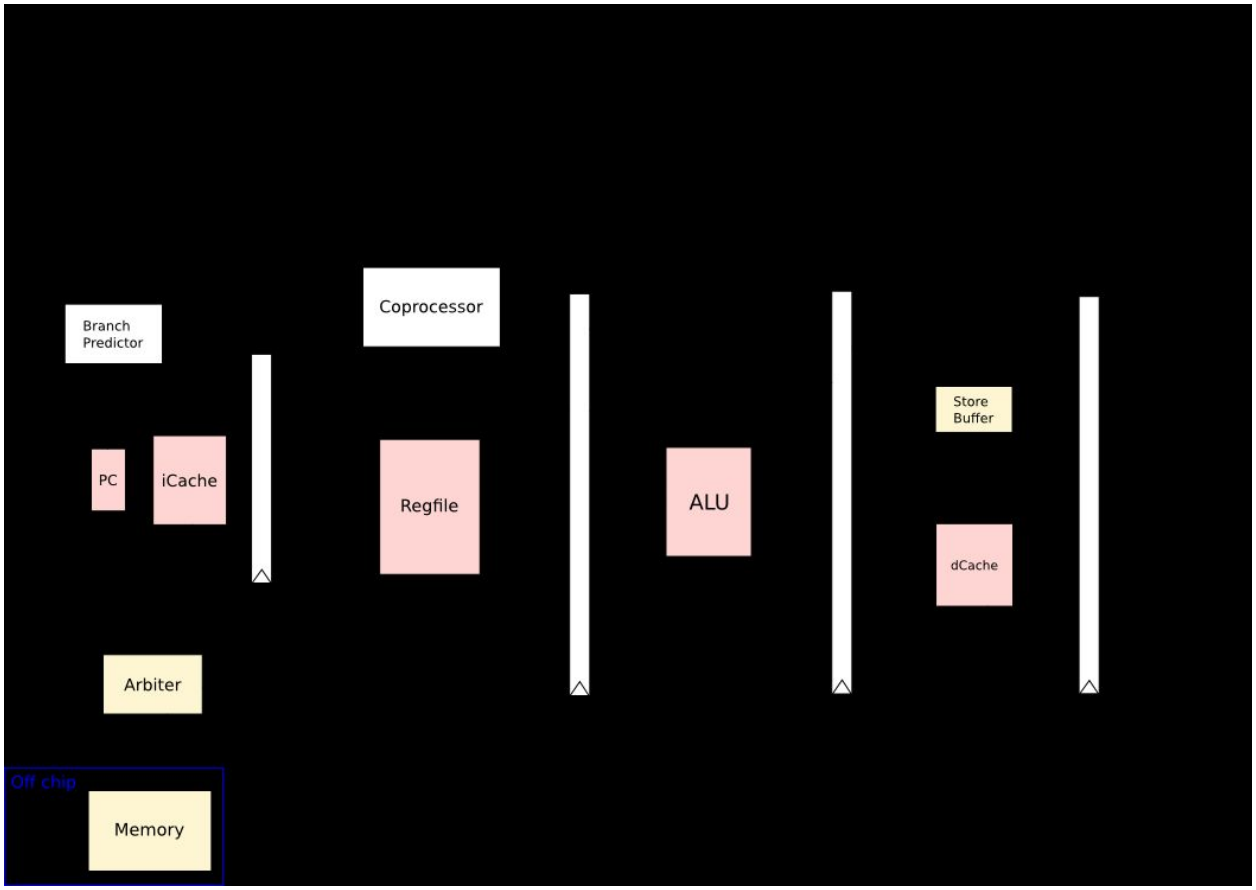
mipsACE

- Features
  - LD
  - ST
  - ADD
  - BRANCH
- Benchmarks
  - Buffer sum
  - Mem copy
  - Matrix multiplication (16x16)
- Supported ISA
- ISA Decoding
- Supported Exceptions
- Supported Syscalls

Features

- Project management through a git repository.
  - Currently, private.
  - Wiki
- MIPS ISA friendly, high compatibility.
  - Syscalls
- Assembly compiler to support the implemented instructions.
- Processor features
  - Fully Pipelined
  - Bypasses
  - Hazard Control
  - Mini OS
  - Exceptions control
  - First level cache

LD



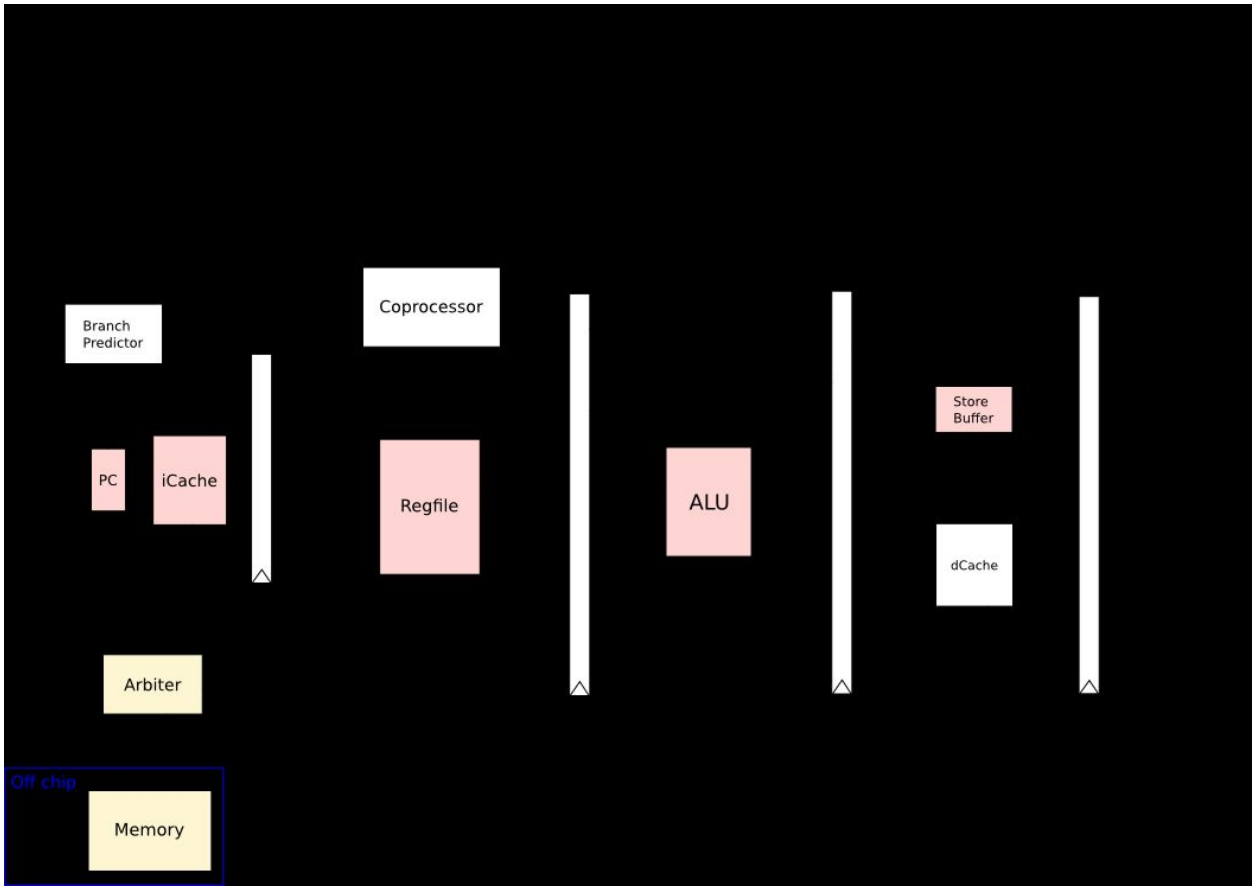
Fetch: Instruction is fetched from Icache, using the address contained in the program counter. If there is a miss there is a stall, and instruction is obtained from memory, being the arbiter an intermediary.

Decode: Load is an I instruction. This way, Rs and Immediate which form the data address are acquired from the register file (bits 25-21) and from the instruction itself (bits 15-0). Control is present in this phase to create signals to command logical units in next phases.

Execute: ALU adds the content of Rs plus the sign-extended Immediate to form the load address.
Memory: Address is searched in the store buffer to know if there has been a recently stored in it. If there has not, a request is sent to Dcache. If data is not in it, another request is sent to main memory, using the arbiter as an intermediary.

WB: Data loaded in the previous phase is stored in the register file, in the register specified by Rt (bits 20-16).

ST



Fetch: Instruction is fetched from Icache, using the address contained in the program counter. If there is a miss there is a stall, and instruction is obtained from memory, being the arbiter an intermediary.

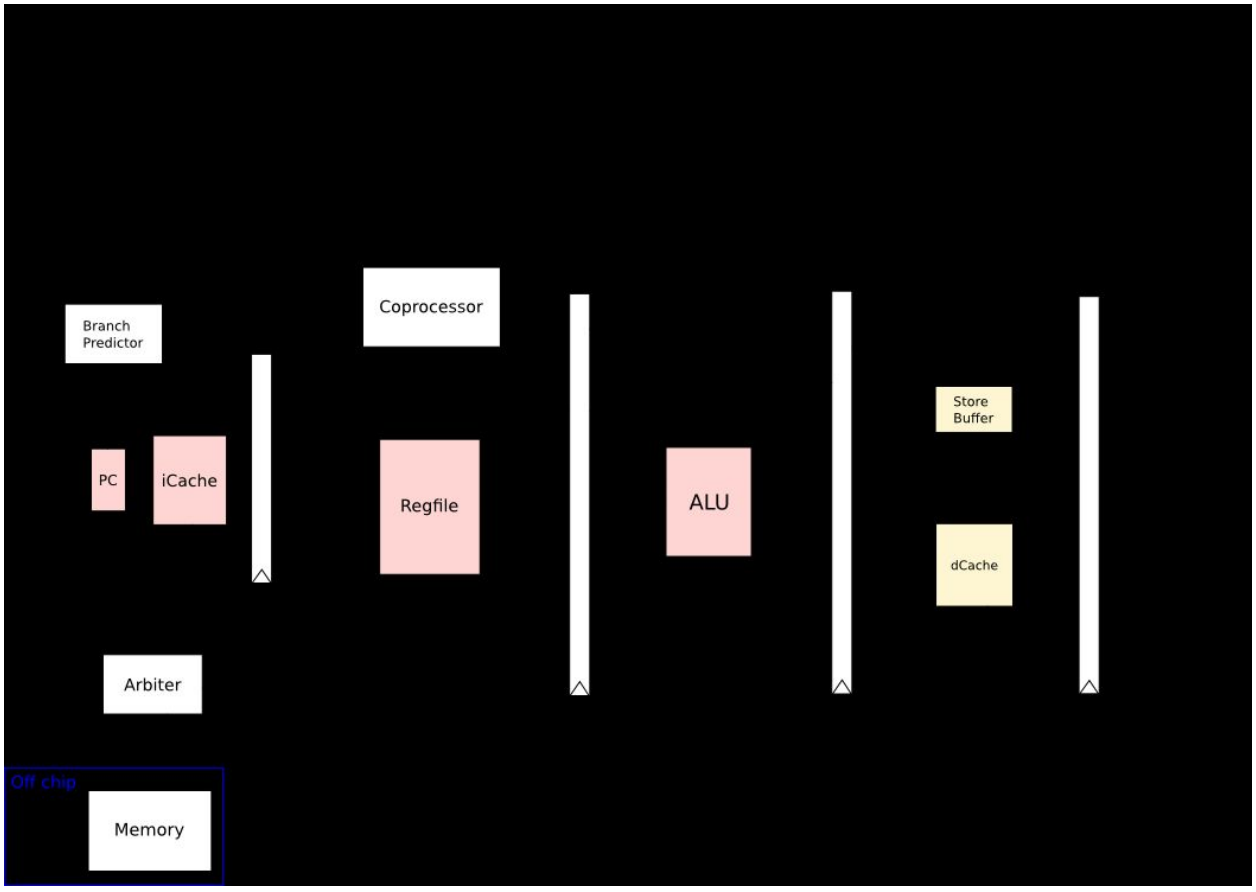
Decode: Store is an I instruction. This way, Rs and Immediate form the address where data is to be stored. They are acquired from the register file (bits 25-21) and from the instruction itself (bits 15-0). Control is present in this phase to create signals to command logical units in next phases. Data contained in register specified in Rt (bits 20-16) is also led to next phases.

Execute: ALU adds the content of Rs plus the sign-extended Immediate to form the store address.

Memory: Address and data are tried to be stored in the store buffer. If it is full, this unit stalls the pipeline and stores up to its capacity in the Dcache, leaving free slots to store this new data. If there is a slot, data and its address are saved in the store buffer. In the future, if there is an ALU operation, their memory phase will be used to store the oldest data in Dcache. If there is a miss, this data will be stored in a future ALU operation or pipeline stall (when full).

WB: logic units in this phase are not used.

ADD

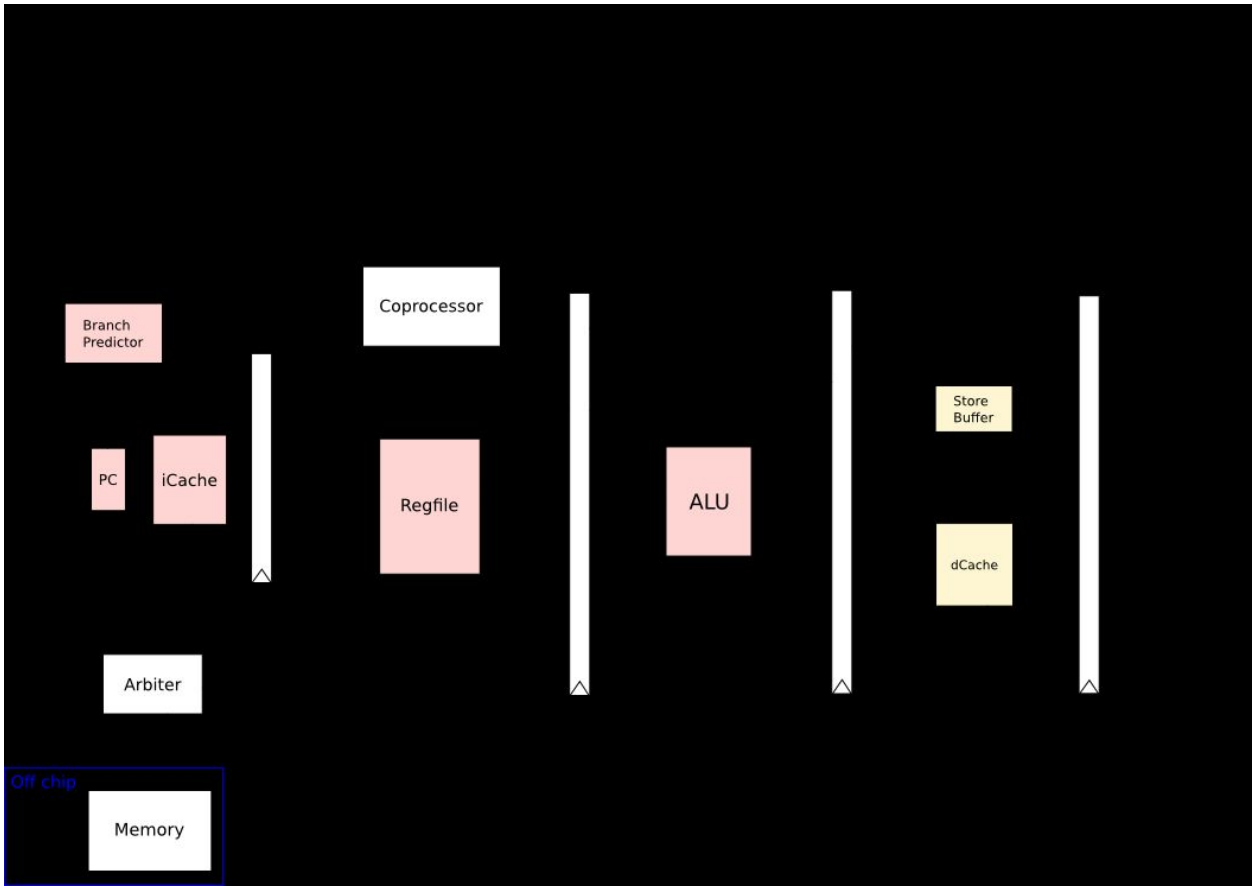


Fetch: Instruction is fetched from Icache, using the address contained in the program counter. If there is a miss there is a stall, and instruction is obtained from memory, being the arbiter an intermediary.

Decode: Add is an R instruction. This way, Rs (bits 25-21) and Rt (bits 20-16) contain the operands of the operation. This data is extracted from the register file and spread to next phases. Control is present in this phase to create signals to command logical units in next phases.

Execute: ALU adds the content of Rs and Rt.
Memory: logic units in this phase are not used.
WB: The summation result from execute phase is stored in the register file, in Rd (bits 15-11).

BRANCH



Fetch: Instruction is fetched from icache, using the address contained in the program counter. If there is a miss there is a stall, and instruction is obtained from memory, being the arbiter an intermediary.

The branch predictor may have an opinion if the address of the instruction is detected as a branch. If the predictor has on record a branch taken state, the next pc will be the pc of the taken branch. If the predictor has on record a state of not taken branch, the next pc will be the pc of the not taken path.

Decode: Branch is an I instruction. This way, Immediate form the address to jump into. It is acquired from the instruction itself (bits 15-0). Control is present in this phase to create signals to command logical units in next phases.

Execute: Immediate is sign-extended and added to the current program counter. PC becomes this new address and next fetch will bring a new instruction from it. Branch predictor collects results from this stage and will update the record of the branch.

Memory: Logic units in this phase are not used.

WB: Logic units in this phase are not used.

Benchmarks

Buffer sum

```
$ ace run buffer_sum.s -dINSTRUMENT -dNO_BPRED
8128
[perf] cycles:                          2761
[perf] instructions:                    1321
[perf] branches:                        281
[perf] branch_misses:                   262
[perf] dcache_loads:                    277
[perf] dcache_load_misses:              131
[perf] dcache_stores:                   322
[perf] dcache_store_misses:             185
[perf] icache_load_misses:              201
[perf] IPC:                        0.478450
```

Buffer sum without branch predictor

```
$ ace run buffer_sum.s -dINSTRUMENT
8128
[perf] cycles:                          2002
[perf] instructions:                    1321
[perf] branches:                        281
[perf] branch_misses:                   4
[perf] dcache_loads:                    277
[perf] dcache_load_misses:              131
[perf] dcache_stores:                   322
[perf] dcache_store_misses:             185
[perf] icache_load_misses:              198
[perf] IPC:                        0.659840
```

Buffer sum with branch predictor

Mem copy

```
$ ace run mem_copy.s -dINSTRUMENT -dNO_BPRED
[perf] cycles:                          3669
[perf] instructions:                    1343
[perf] branches:                        263
[perf] branch_misses:                   256
[perf] dcache_loads:                    909
[perf] dcache_load_misses:              777
[perf] dcache_stores:                   835
[perf] dcache_store_misses:             576
[perf] icache_load_misses:              78
[perf] IPC:                        0.366040
```

Mem copy without branch predictor

```
$ ace run mem_copy.s -dINSTRUMENT
[perf] cycles:                          2919
[perf] instructions:                    1343
[perf] branches:                        263
[perf] branch_misses:                   3
[perf] dcache_loads:                    909
[perf] dcache_load_misses:              777
[perf] dcache_stores:                   835
[perf] dcache_store_misses:             576
[perf] icache_load_misses:              78
[perf] IPC:                        0.460089
```

Mem copy with branch predictor

Matrix multiplication (16x16)

```
$ ace run matrix_multiply.s -dINSTRUMENT -dNO_BPRED
[perf] cycles:                          7273
[perf] instructions:                    3883
[perf] branches:                        295
[perf] branch_misses:                   257
[perf] dcache_loads:                    1722
[perf] dcache_load_misses:              1206
[perf] dcache_stores:                   70
[perf] dcache_store_misses:             51
[perf] icache_load_misses:              1886
[perf] IPC:                        0.533892
```

Mem copy without branch predictor

```
$ ace run matrix_multiply.s -dINSTRUMENT
[perf] cycles:                          5848
[perf] instructions:                    3883
[perf] branches:                        295
[perf] branch_misses:                   18
[perf] dcache_loads:                    1722
[perf] dcache_load_misses:              1206
[perf] dcache_stores:                   70
[perf] dcache_store_misses:             51
[perf] icache_load_misses:              455
[perf] IPC:                        0.663988
```

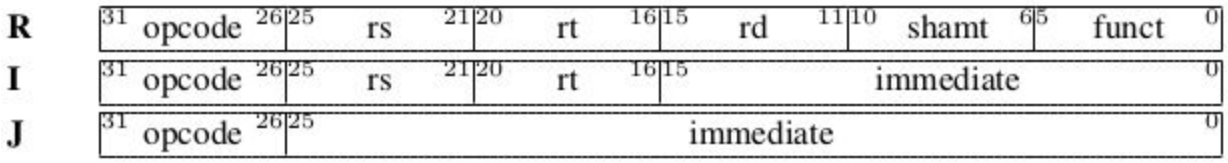
Mem copy with branch predictor

Supported ISA

INSTRUCTION	TYPE	PRIVILEGED
add	R	NO
addi	I	NO
sub	R	NO
and	R	NO
andi	I	NO
or	R	NO
ori	I	NO
xor	R	NO
xori	I	NO
sll	R	NO
srl	R	NO
sra	R	NO
slt	R	NO
beq	I	NO
bne	I	NO
j	J	NO
jal	J	NO
jr	R	NO
move	P	NO
lb	I	NO
lui	I	NO
lw	I	NO

li	P	NO
la	P	NO
sb	I	NO
sw	I	NO
syscall	R	NO
mfc0	I	YES
mtc0	I	YES
mul	R	NO
div	R	NO
eret	I	YES
nop	P	NO

ISA Decoding



Supported Exceptions

Number	Function
0	Int Interrupt (hardware)
4	AdEL Address Error Exception (load or instruction fetch)
5	AdES Address Error Exception (store)

6	IBE Bus Error on Instruction Fetch
7	DBE Bus Error on Load or Store
8	Sys Syscall Exception
9	Bp Breakpoint Exception
10	RI Reserved Instruction Exception
11	CpU Coprocessor Unimplemented
12	Ov Arithmetic Overflow Exception
13	Tr Trap

Supported Syscalls

Number	Function
0	Print integer as hex value
1	Print integer, $a0 = value
4	Print string, $a0 = address of string
5	Read integer, $v0 = value read
10	Exit, end of program
11	Print character, $a0 = character
12	Read character, $v0 = character read