Computation structures

INFO0012

Fall 2020

Outline

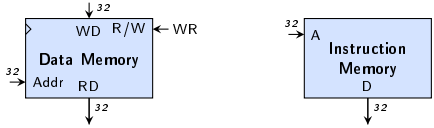
Outline

Outline

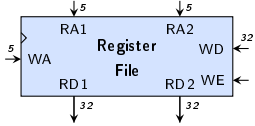
Components

High level components

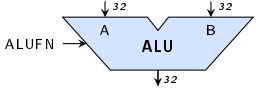
- memories



- register file



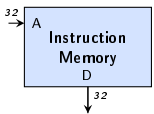
- ALU



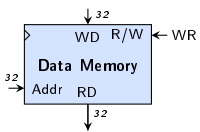
- multiplexers, gates...

Memories

- For pedagogical reasons: distinct memory for instructions and data (Harvard architecture)
- Instruction memory



  - Address (A) on 32 bits (multiple of 4, 2 lsb are 0)
  - Output (D) on 32 bits
- Data memory

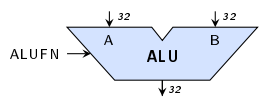


  - Address (Adr) on 32 bits (multiple of 4, 2 lsb are 0)
  - Output (RD) on 32 bits
  - If WR is 1, writing data (WD) at address (Adr) on the next clock trigger (when the clock changes from 0 to 1)

ALU (reminder)

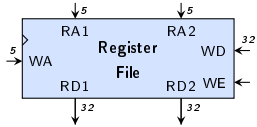
- Arithmetic operations on integers (addition, subtraction, possibly multiplication and division)
- Bit-wise logic operations (and, or, xor)
- Shift operations
- Arithmetic Logic Unit

A and B arguments
ALUFN specifies the operation



Register file

- $2^5 - 1$ registers
- Ability to read two registers simultaneously
- Ability to write in one register
- Register 31 is constant 0 (on 32 bits)



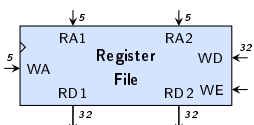
Inputs/outputs of the register file

- $RA1^5$ (in) first register address for reading
- $RA2^5$ (in) second register address for reading
- WA^5 (in) register address for writing
- WD^{32} (in) data to write in register addressed by WA
- WE^1 (in) enable writing (1) or not (0), at the next clock trigger
- $RD1^{32}$ (out) data in register addressed by RA1
- $RD2^{32}$ (out) data in register addressed by RA2

Register file (2)

Simplified implementation

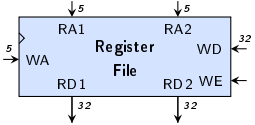
- only registers 0, 1, 2, 3, 4, 29, 30, and 31 are explicitly present

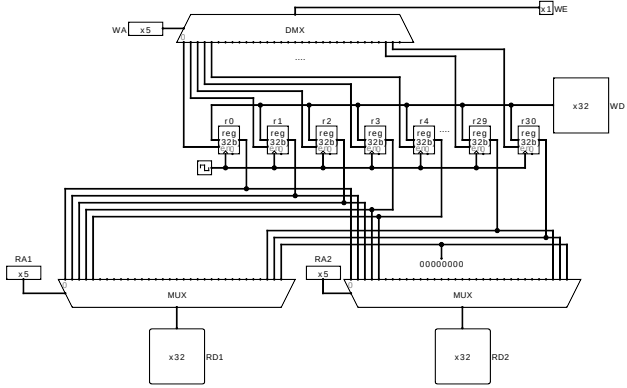


Register file (2)

Simplified implementation

- only registers 0, 1, 2, 3, 4, 29, 30, and 31 are explicitly present





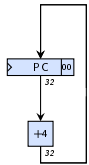
Outline

Reading and decoding instructions

- 32 bits program counter PC

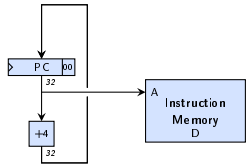
Reading and decoding instructions

- 32 bits program counter PC
- Gets incremented by 4 at each clock trigger



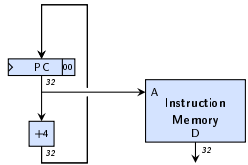
Reading and decoding instructions

- 32 bits program counter PC
- Gets incremented by 4 at each clock trigger
- PC serves as address for instruction memory



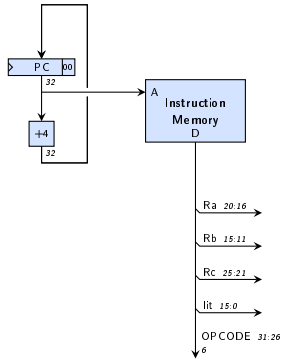
Reading and decoding instructions

- 32 bits program counter PC
- Gets incremented by 4 at each clock trigger
- PC serves as address for instruction memory
- Instruction memory provides word (32 bits) at given address



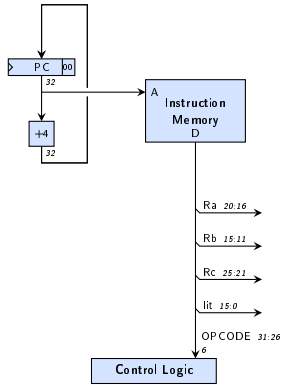
Reading and decoding instructions

- 32 bits program counter PC
- Gets incremented by 4 at each clock trigger
- PC serves as address for instruction memory
- Instruction memory provides word (32 bits) at given address
- Instruction is decoded (cut into components)



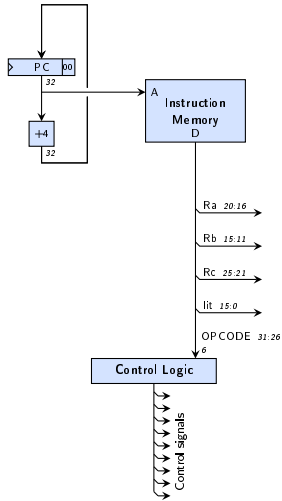
Reading and decoding instructions

- 32 bits program counter PC
- Gets incremented by 4 at each clock trigger
- PC serves as address for instruction memory
- Instruction memory provides word (32 bits) at given address
- Instruction is decoded (cut into components)
- The opcode is the input address of the control logic (ROM)



Reading and decoding instructions

- 32 bits program counter PC
- Gets incremented by 4 at each clock trigger
- PC serves as address for instruction memory
- Instruction memory provides word (32 bits) at given address
- Instruction is decoded (cut into components)
- The opcode is the input address of the control logic (ROM)
- The control logic will provide signals to enable subcircuits and route information in the CPU

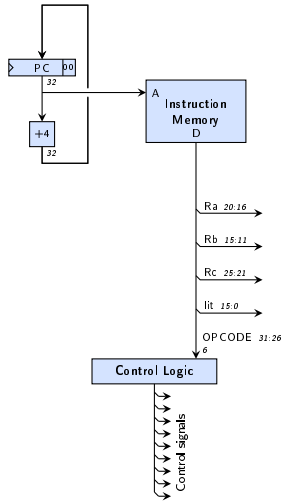


Reading and decoding instructions (2)

- PC increases by steps of 4 at each clock trigger
  (that is, every time the clock goes from 0 to 1)
- The least two significant bits of PC are always 0
- PC is used as address for the instruction memory
- The memory output can be decomposed in register addresses (5 bits): Ra, Rb, Rc
- If necessary, the constant embedded in the instruction can also be recovered; here it is converted from 16 bits to 32 bits implicitly with signed extension
- The OPCODE is extracted by taking the 6 most significant bits
- The OPCODE is given to the control logic, which takes care of activating certain control signals

ALU instructions (before)

```
ADD(Ra,Rb,Rc):
    PC ⟵ PC + 4
    Reg[Rc] ⟵ Reg[Ra] + Reg[Rb]
```



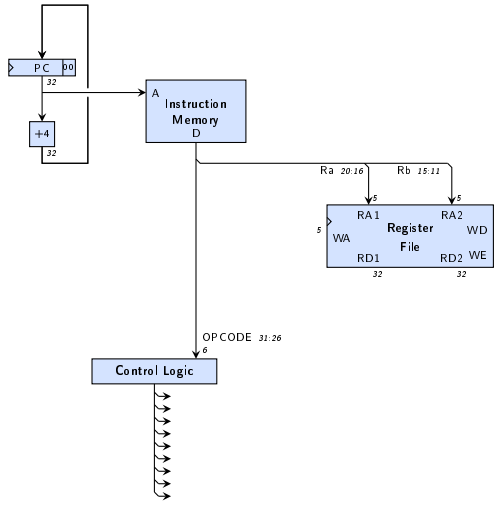
ALU instructions

`ADD(Ra,Rb,Rc)`:

PC ⟵ PC + 4

Reg[Rc] ⟵ Reg[Ra] + Reg[Rb]

- Bits 20:16 and 15:11 given as two read addresses to the register file



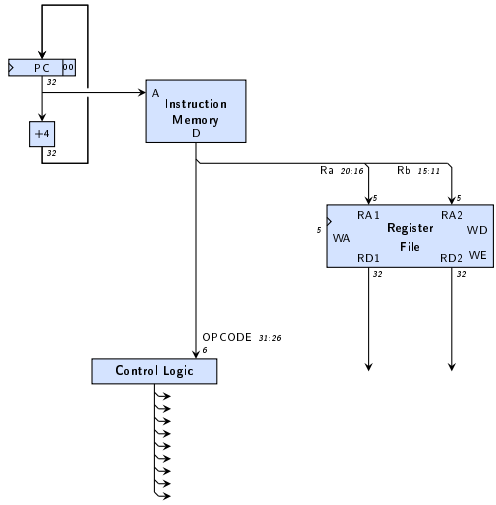
ALU instructions

`ADD(Ra,Rb,Rc)`:

PC ⟵ PC + 4

Reg[Rc] ⟵ Reg[Ra] + Reg[Rb]

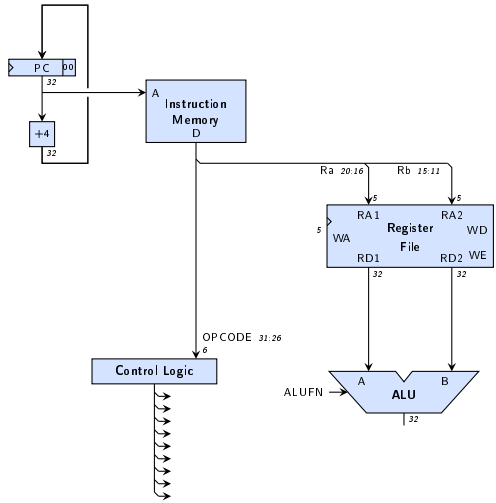
- Bits 20:16 and 15:11 given as two read addresses to the register file
- The register file outputs the contents of the registers



ALU instructions

```
ADD(Ra,Rb,Rc):
    PC ⟵ PC + 4
    Reg[Rc] ⟵ Reg[Ra] + Reg[Rb]
```

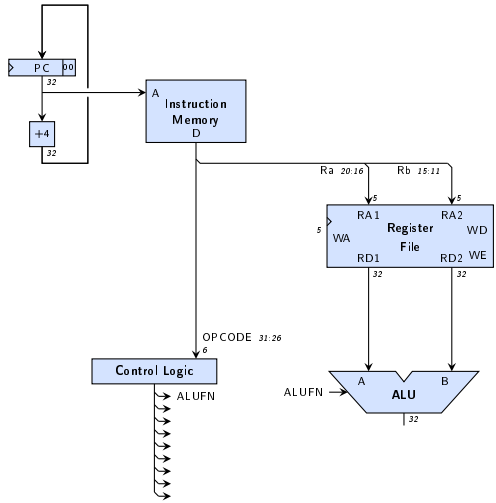
- Bits 20:16 and 15:11 given as two read addresses to the register file
- The register file outputs the contents of the registers
- These are given as arguments to the ALU



ALU instructions

```
ADD(Ra,Rb,Rc):
    PC ⟵ PC + 4
    Reg[Rc] ⟵ Reg[Ra] + Reg[Rb]
```

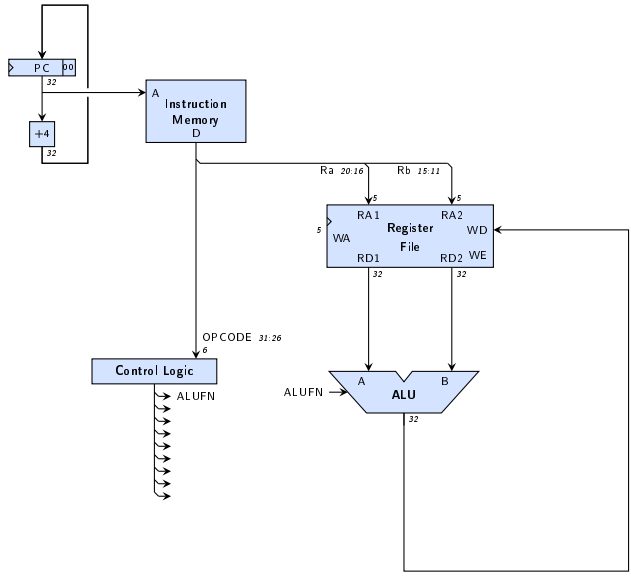
- Bits 20:16 and 15:11 given as two read addresses to the register file
- The register file outputs the contents of the registers
- These are given as arguments to the ALU
- The control logic ensures the ALU computes the right operation



ALU instructions

```
ADD(Ra,Rb,Rc):
    PC ⟵ PC + 4
    Reg[Rc] ⟵ Reg[Ra] + Reg[Rb]
```

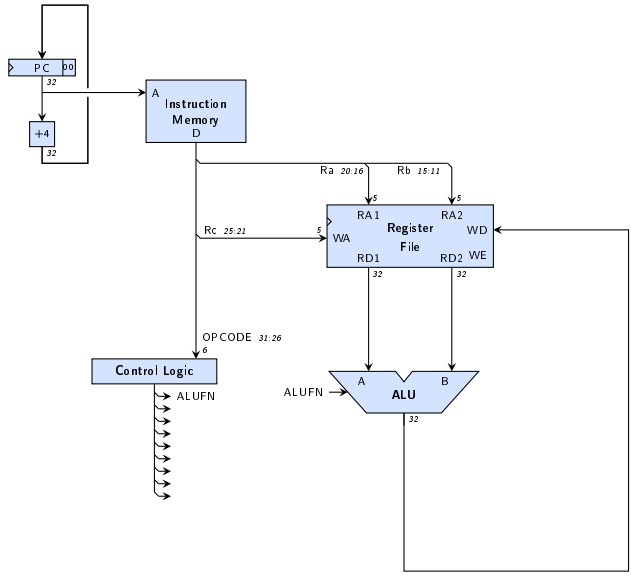
- Bits 20:16 and 15:11 given as two read addresses to the register file
- The register file outputs the contents of the registers
- These are given as arguments to the ALU
- The control logic ensures the ALU computes the right operation
- The result is given as data for writing at the next clock trigger



ALU instructions

```
ADD(Ra,Rb,Rc):
    PC ⟵ PC + 4
    Reg[Rc] ⟵ Reg[Ra] + Reg[Rb]
```

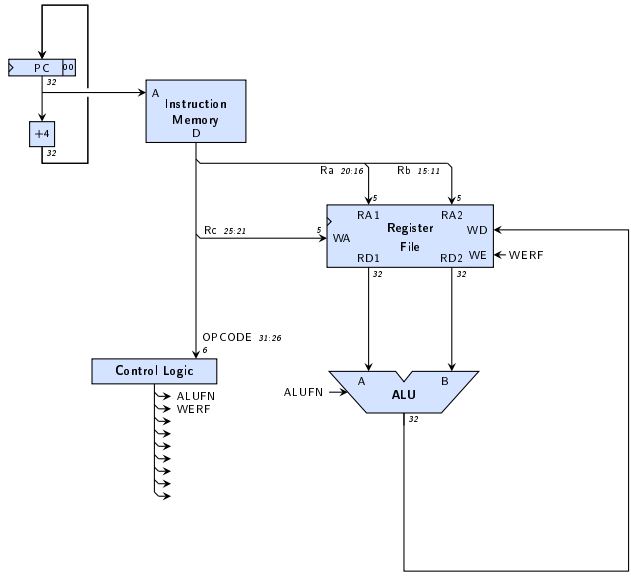
- Bits 20:16 and 15:11 given as two read addresses to the register file
- The register file outputs the contents of the registers
- These are given as arguments to the ALU
- The control logic ensures the ALU computes the right operation
- The result is given as data for writing at the next clock trigger
- Bits 25:21 serve as write address for the register file



ALU instructions (after)

```
ADD(Ra,Rb,Rc):
    PC ⟵ PC + 4
    Reg[Rc] ⟵ Reg[Ra] + Reg[Rb]
```

- Bits 20:16 and 15:11 given as two read addresses to the register file
- The register file outputs the contents of the registers
- These are given as arguments to the ALU
- The control logic ensures the ALU computes the right operation
- The result is given as data for writing at the next clock trigger
- Bits 25:21 serve as write address for the register file
- The control logic notifies the register file to write data



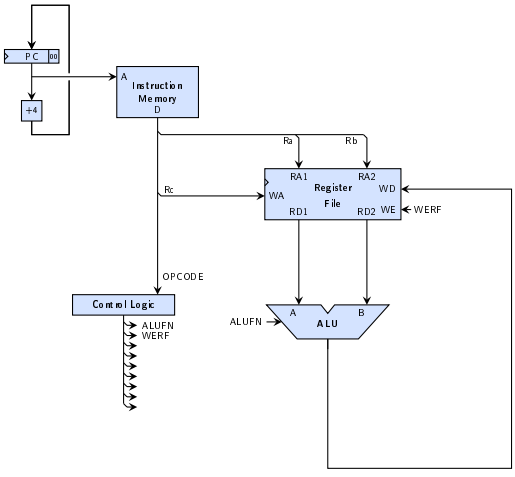
ALU instructions (description)

- The instruction specifies three registers, two arguments and the destination, on which the operation works.
- Depending on the OPCODE, the control logic circuit selects the operation that the ALU will do (or rather, the result that should come out of the ALU)

  ALUFN^4 operation done by the ALU ($+$, $-$, $*$, $/$, $\wedge$, $\vee$, $\otimes$, $=$, $<$, $\leq$, $\ll$, $\gg$, $\gg'$)

  WERF^1 writing in a register (1)

ALU instructions (control logic)



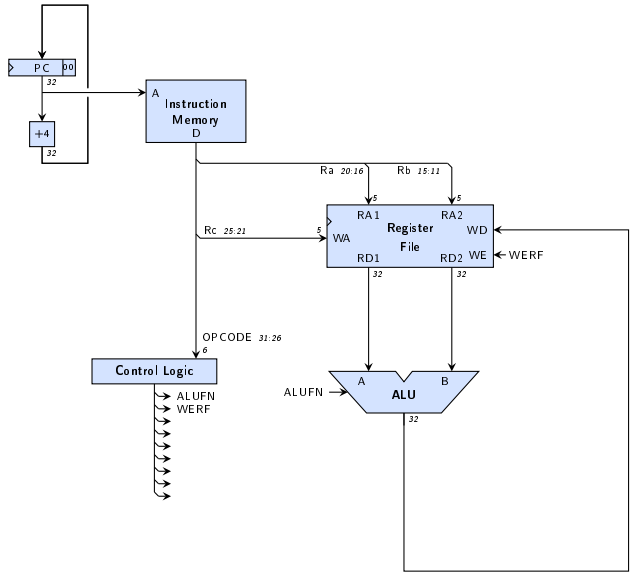
	ALUFN	WERF
OP	op.	1

ALU instructions with constants (before)

ADDC(Ra,literal,Rc):

PC ⟵ PC + 4

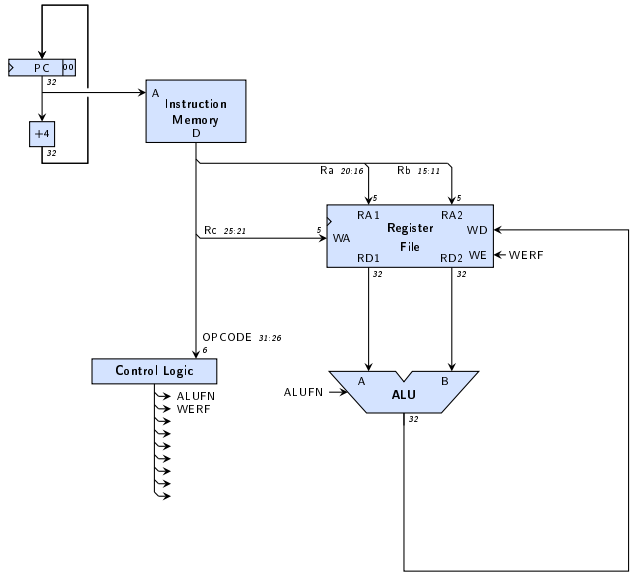
Reg[Rc] ⟵ Reg[Ra] + SXT(literal)



ALU instructions with constants (before)

ADDC(Ra,literal,Rc):
PC ⟵ PC + 4
Reg[Rc] ⟵ Reg[Ra] + SXT(literal)

- Bits 20:16 given as read address to the register file



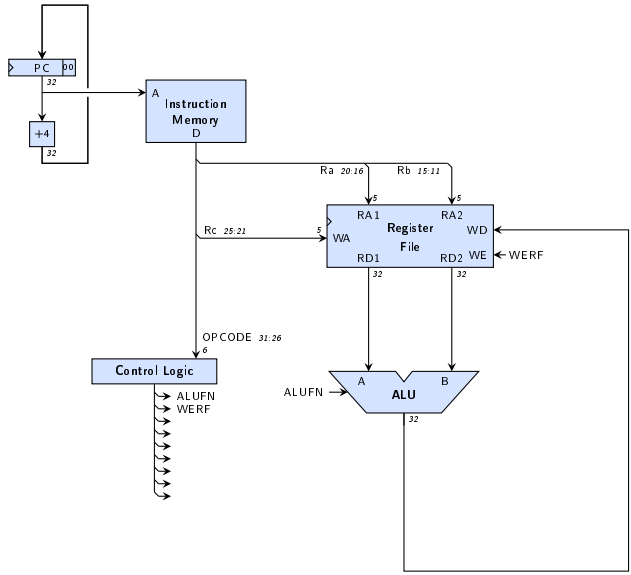
ALU instructions with constants (before)

ADDC(Ra,literal,Rc):

PC ⟵ PC + 4

Reg[Rc] ⟵ Reg[Ra] + SXT(literal)

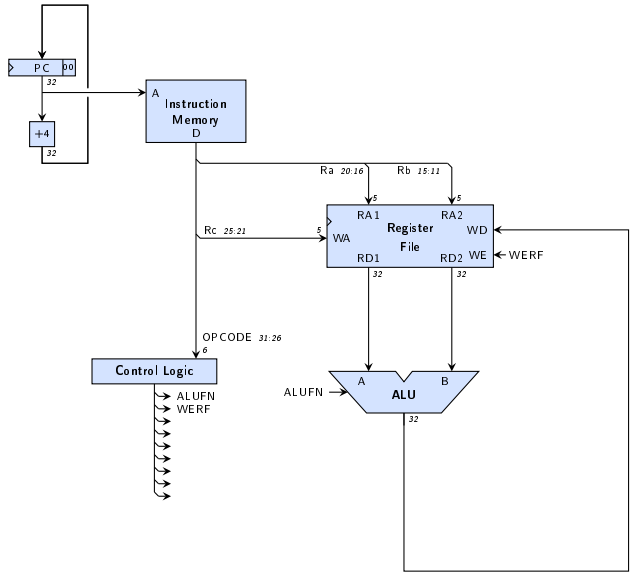
- Bits 20:16 given as read address to the register file
- The register file outputs the contents of the registers



ALU instructions with constants (before)

ADDC(Ra,literal,Rc):
PC ⟵ PC + 4
Reg[Rc] ⟵ Reg[Ra] + SXT(literal)

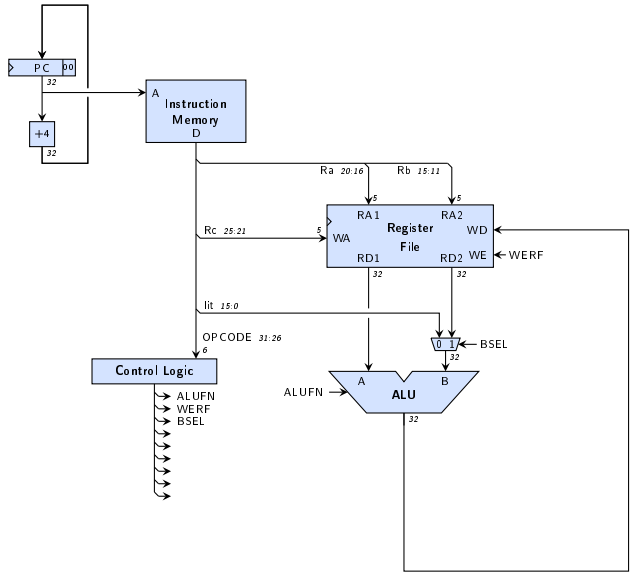
- Bits 20:16 given as read address to the register file
- The register file outputs the contents of the registers
- Value of second register is ignored



ALU instructions with constants (after)

```
ADDC(Ra,literal,Rc):
    PC ⟵ PC + 4
    Reg[Rc]  ⟵ Reg[Ra] + SXT(literal)
```

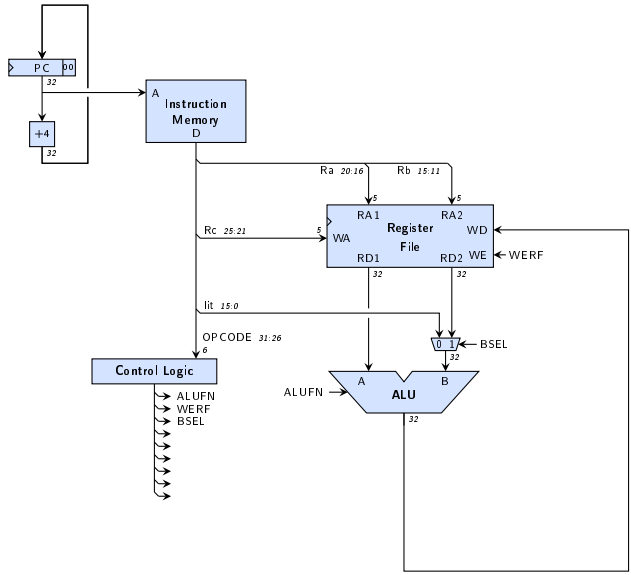
- Bits 20:16 given as read address to the register file
- The register file outputs the contents of the registers
- Value of second register is ignored
- First register and signed extension of constant given as arguments to the ALU



ALU instructions with constants (after)

```
ADDC(Ra,literal,Rc):
    PC ⟵ PC + 4
    Reg[Rc]  ⟵ Reg[Ra] + SXT(literal)
```

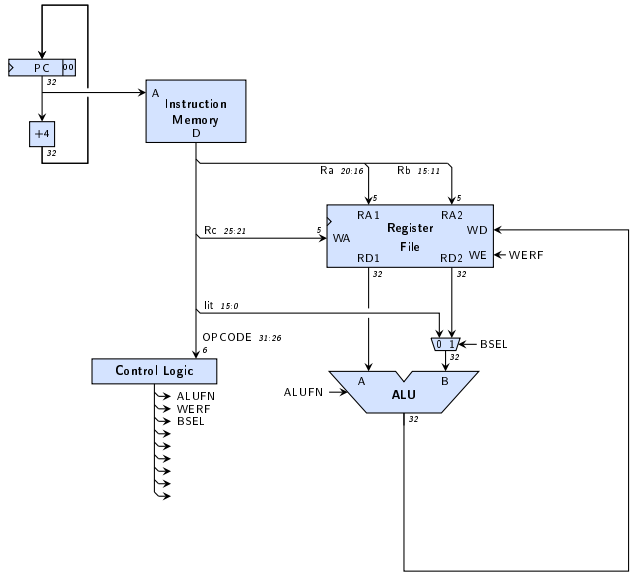
- Bits 20:16 given as read address to the register file
- The register file outputs the contents of the registers
- Value of second register is ignored
- First register and signed extension of constant given as arguments to the ALU
- The result is given as data for writing at the next clock trigger



ALU instructions with constants (after)

ADDC(Ra,literal,Rc):
PC ⟵ PC + 4
Reg[Rc] ⟵ Reg[Ra] + SXT(literal)

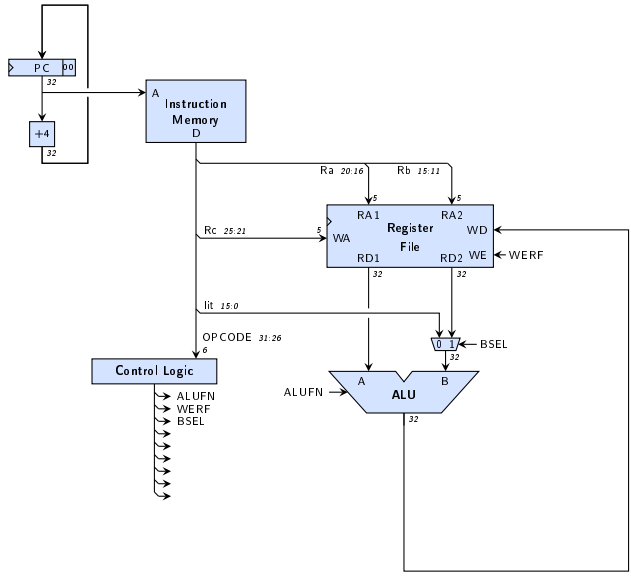
- Bits 20:16 given as read address to the register file
- The register file outputs the contents of the registers
- Value of second register is ignored
- First register and signed extension of constant given as arguments to the ALU
- The result is given as data for writing at the next clock trigger
- Bits 25:21: register file write address



ALU instructions with constants (after)

```
ADDC(Ra,literal,Rc):
    PC ⟵ PC + 4
    Reg[Rc]  ⟵ Reg[Ra] + SXT(literal)
```

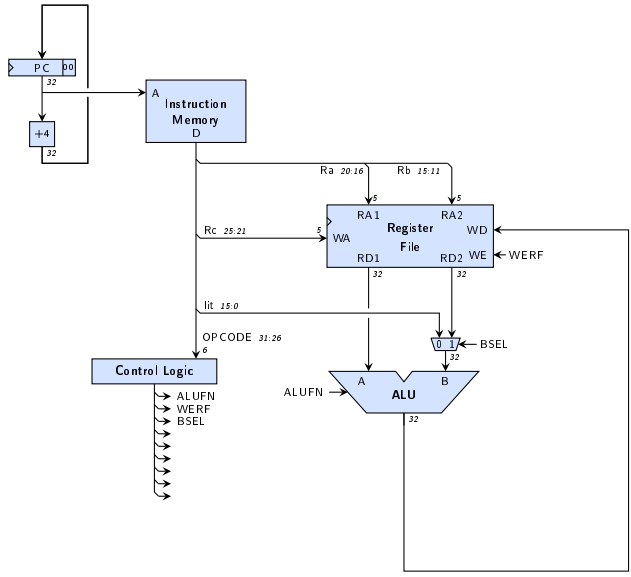
- Bits 20:16 given as read address to the register file
- The register file outputs the contents of the registers
- Value of second register is ignored
- First register and signed extension of constant given as arguments to the ALU
- The result is given as data for writing at the next clock trigger
- Bits 25:21: register file write address
- The control logic notifies the register file to write data



ALU instructions with constants (after)

```
ADDC(Ra,literal,Rc):
    PC ⟵ PC + 4
    Reg[Rc] ⟵ Reg[Ra] + SXT(literal)
```

- Bits 20:16 given as read address to the register file
- The register file outputs the contents of the registers
- Value of second register is ignored
- First register and signed extension of constant given as arguments to the ALU
- The result is given as data for writing at the next clock trigger
- Bits 25:21: register file write address
- The control logic notifies the register file to write data
- The control logic ensures the ALU computes the right operation

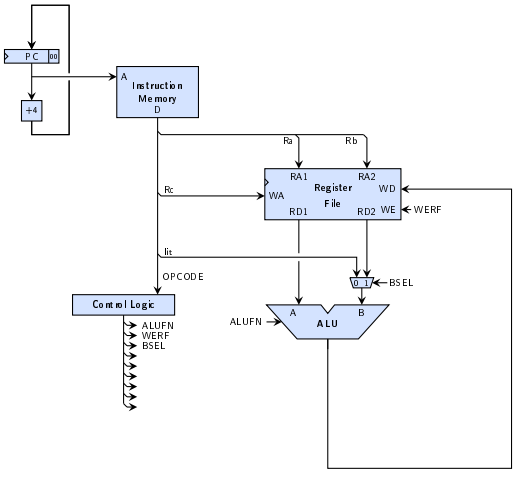


ALU instructions with constants (description)

- Depending on the OPCODE, the control logic selects (the operation for the ALU and) whether the second argument is a register or a constant (BSEL)

  BSEL[1] embedded constant in instruction (0) or register (1) for the B ALU argument

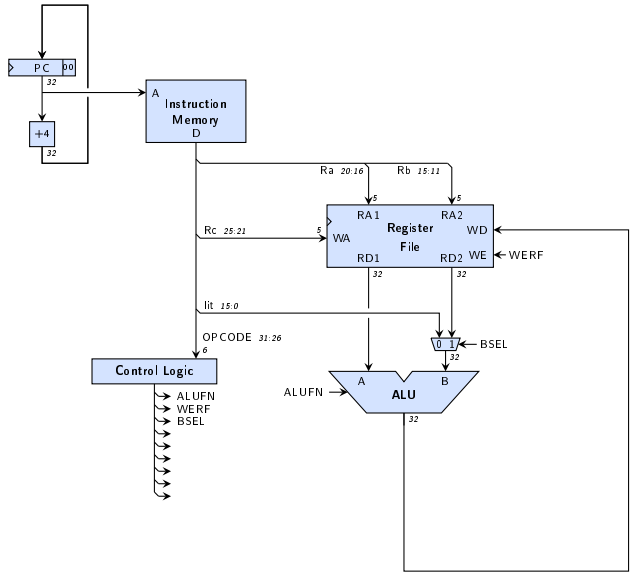
ALU instructions with constants (control logic)



	ALUFN	WERF	BSEL
OP	op.	1	1
OPC	op.	1	0

Memory instruction LD (before)

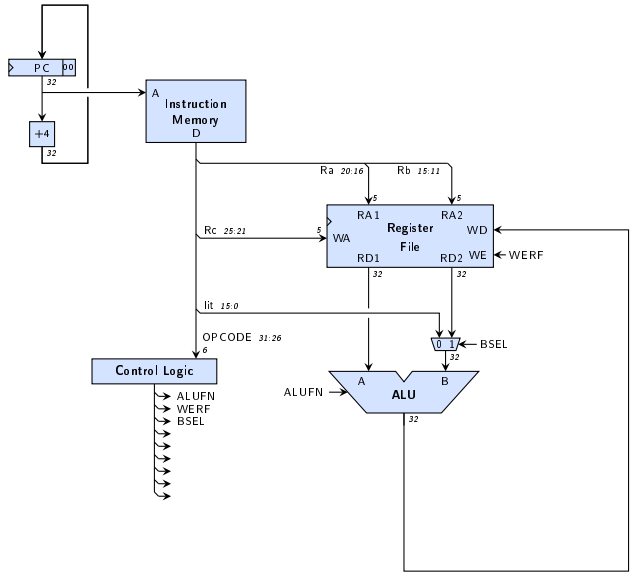
LD(Ra, literal, Rc):
PC ⟵ PC + 4
Reg[Rc] ⟵ Mem[Reg[Ra] + SXT(literal)]



Memory instruction LD (before)

LD(Ra, literal, Rc):
PC ⟵ PC + 4
Reg[Rc] ⟵ Mem[Reg[Ra] + SXT(literal)]

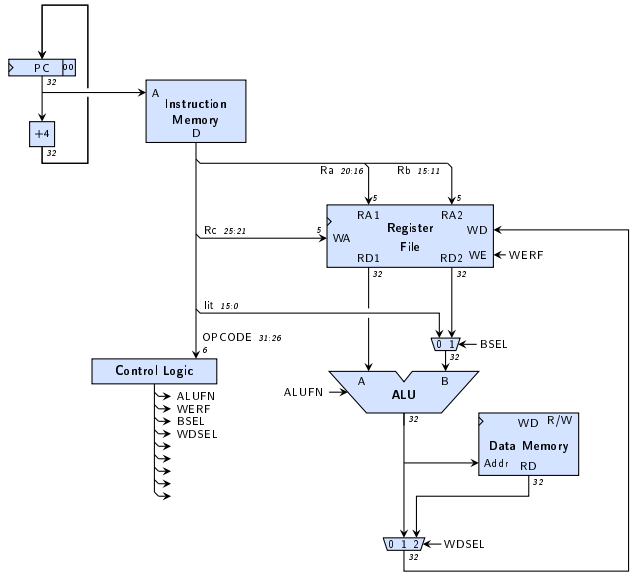
- Essentially as for ADDC
- The control logic ensures the ALU computes an addition



Memory instruction LD (after)

LD(Ra, literal, Rc):
PC ⟵ PC + 4
Reg[Rc] ⟵ Mem[Reg[Ra] + SXT(literal)]

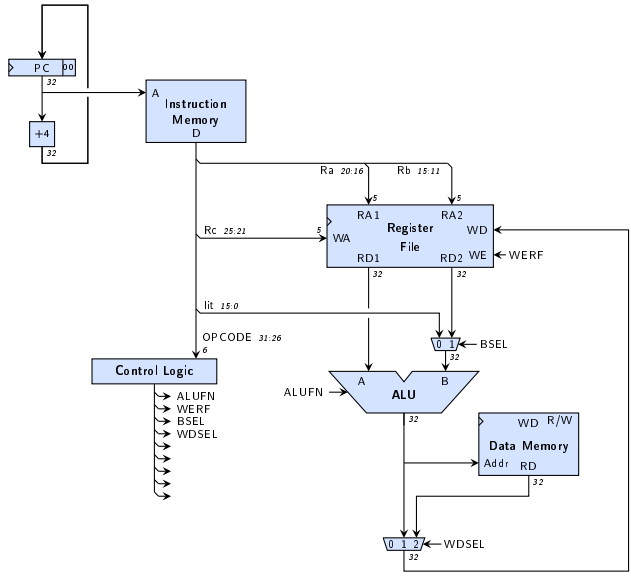
- Essentially as for ADDC
- The control logic ensures the ALU computes an addition
- The result of the ALU is given as address to memory



Memory instruction LD (after)

LD(Ra, literal, Rc):
PC ⟵ PC + 4
Reg[Rc] ⟵ Mem[Reg[Ra] + SXT(literal)]

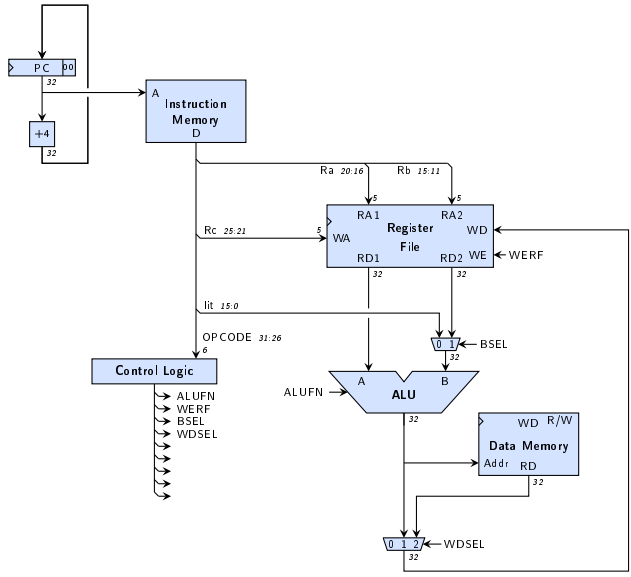
- Essentially as for ADDC
- The control logic ensures the ALU computes an addition
- The result of the ALU is given as address to memory
- The data from memory is given as data for writing at the next clock trigger



Memory instruction LD (after)

LD(Ra, literal, Rc):
PC ⟵ PC + 4
Reg[Rc] ⟵ Mem[Reg[Ra] + SXT(literal)]

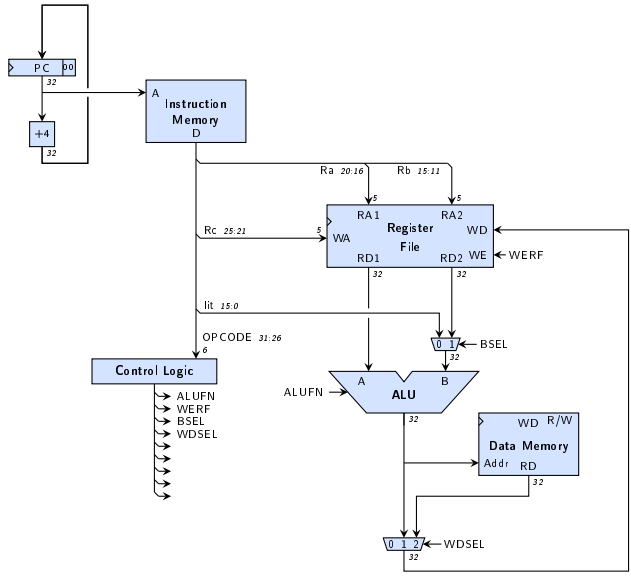
- Essentially as for ADDC
- The control logic ensures the ALU computes an addition
- The result of the ALU is given as address to memory
- The data from memory is given as data for writing at the next clock trigger
- Bits 25:21: register file write address



Memory instruction LD (after)

LD(Ra, literal, Rc):
PC ⟵ PC + 4
Reg[Rc] ⟵ Mem[Reg[Ra] + SXT(literal)]

- Essentially as for ADDC
- The control logic ensures the ALU computes an addition
- The result of the ALU is given as address to memory
- The data from memory is given as data for writing at the next clock trigger
- Bits 25:21: register file write address
- The control logic notifies the register file to write data



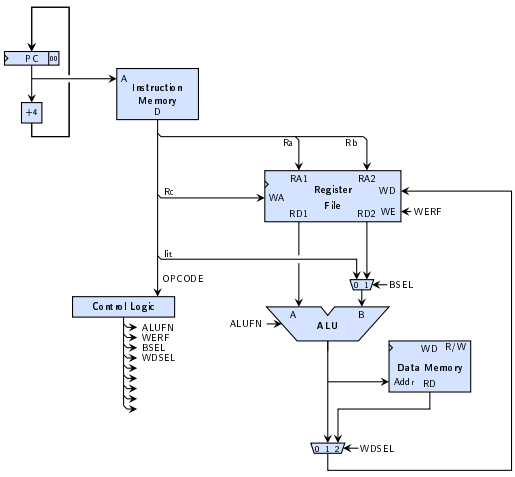
Memory instruction LD (description)

If the OPCODE corresponds to the LD instruction, the control logic will

- select the ALU operation so that its output corresponds to the reading address in memory
- route the read data towards the register file, so that the data is stored on the next clock trigger

$WDSEL^2$ selects either the ALU result (1), or the read data from the memory (2) as write data for the register file
Option (0) will be used later

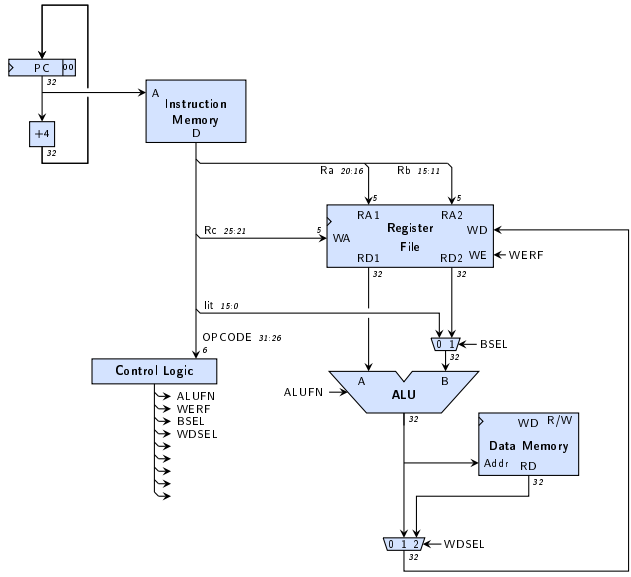
Memory instruction LD (control logic)



	ALUFN	WERF	BSEL	WDSEL
OP	op.	1	1	1
OPC	op.	1	0	1
LD	+	1	0	2

Memory instruction ST (before)

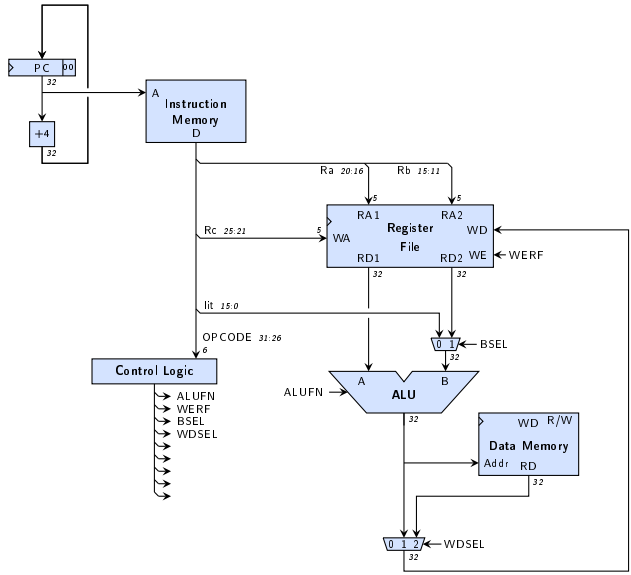
ST(Rc, literal, Ra)
PC ⟵ PC + 4
Mem[Reg[Ra] + SXT(literal)] ⟵ Reg[Rc]



Memory instruction ST (before)

ST(Rc, literal, Ra)
PC ⟵ PC + 4
Mem[Reg[Ra] + SXT(literal)] ⟵ Reg[Rc]

- Essentially as for LD



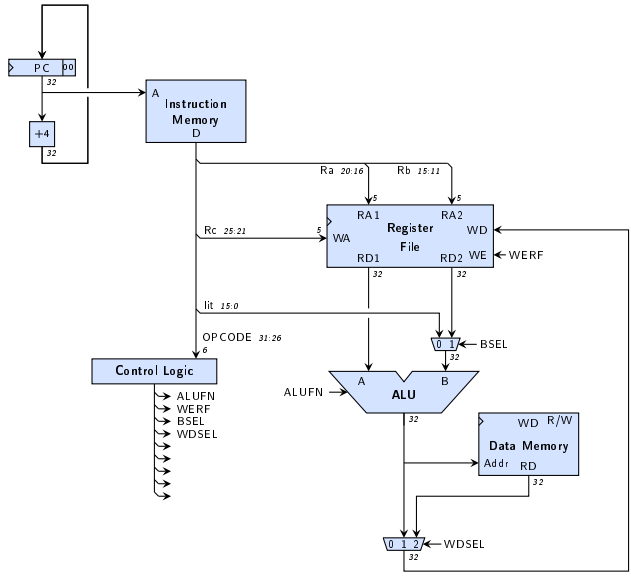
Memory instruction ST (before)

`ST(Rc,literal,Ra)`

PC ⟵ PC + 4

Mem[Reg[Ra] + SXT(literal)] ⟵ Reg[Rc]

- Essentially as for `LD`
- The result of the ALU is given as address to memory



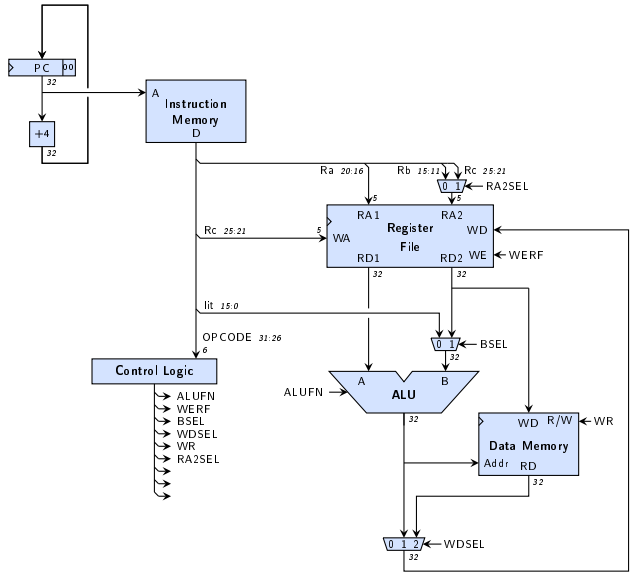
Memory instruction ST (after)

ST(Rc, literal, Ra)

PC ⟵ PC + 4

Mem[Reg[Ra] + SXT(literal)] ⟵ Reg[Rc]

- Essentially as for LD
- The result of the ALU is given as address to memory
- The second output of the register file given as data to memory for writing at the next clock trigger



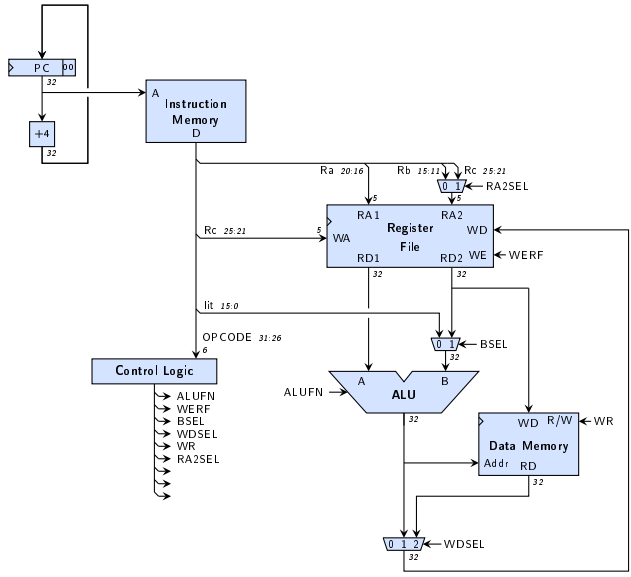
Memory instruction ST (after)

`ST(Rc,literal,Ra)`

PC ⟵ PC + 4

Mem[Reg[Ra] + SXT(literal)] ⟵ Reg[Rc]

- Essentially as for `LD`
- The result of the ALU is given as address to memory
- The second output of the register file given as data to memory for writing at the next clock trigger
- Bits 25:21 is the second register file read address!



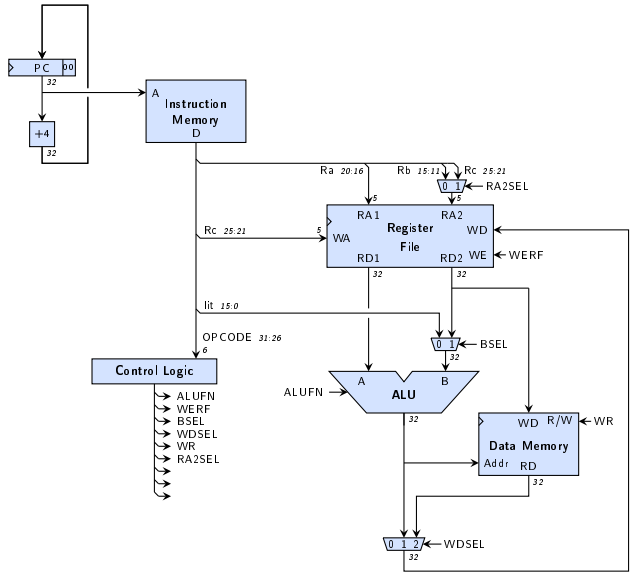
Memory instruction ST (after)

`ST(Rc,literal,Ra)`

PC ⟵ PC + 4

Mem[Reg[Ra] + SXT(literal)] ⟵ Reg[Rc]

- Essentially as for `LD`
- The result of the ALU is given as address to memory
- The second output of the register file given as data to memory for writing at the next clock trigger
- Bits 25:21 is the second register file read address!
- The control logic notifies the memory to write data



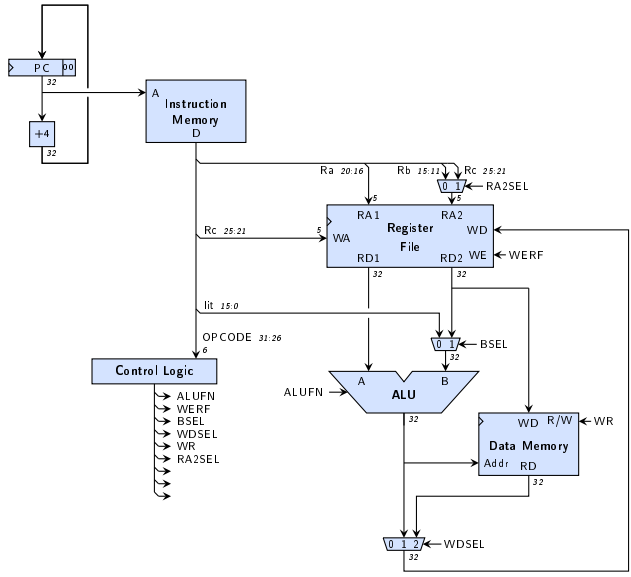
Memory instruction ST (after)

ST(Rc, literal, Ra)

PC ⟵ PC + 4

Mem[Reg[Ra] + SXT(literal)] ⟵ Reg[Rc]

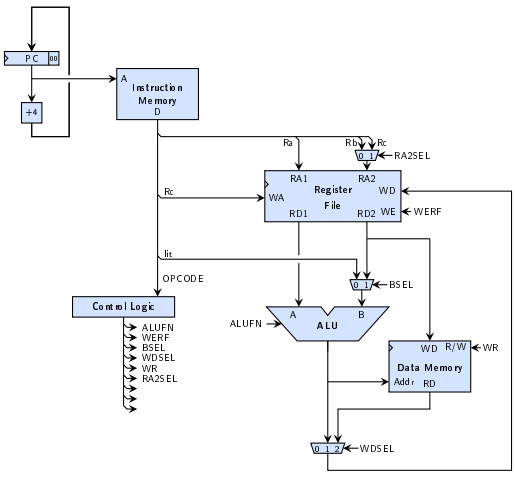
- Essentially as for LD
- The result of the ALU is given as address to memory
- The second output of the register file given as data to memory for writing at the next clock trigger
- Bits 25:21 is the second register file read address!
- The control logic notifies the memory to write data
- The register file will not store data at the next clock trigger



Memory instruction ST (description)

WR[1]	is (1) if data is stored in memory (at the next clock trigger)
RA2SEL[1]	selects either instruction bits 15:11 or bits 25:21 as second register address

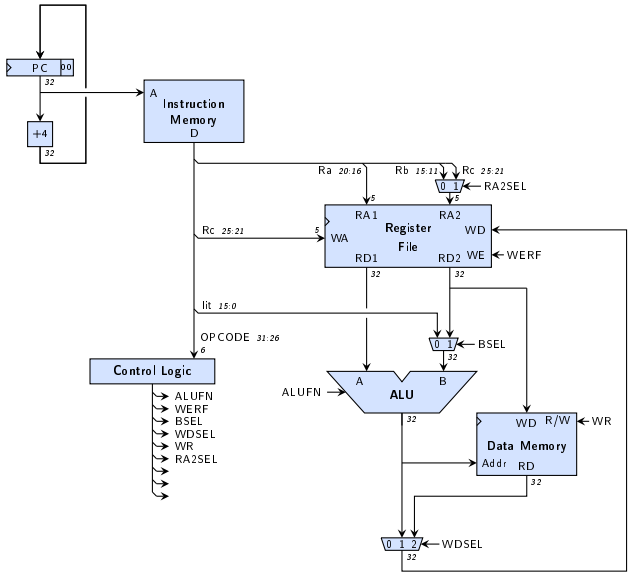
Memory instruction ST (control logic)



	ALUFN	WERF	BSEL	WDSEL	WR	RA2SEL
OP	op.	1	1	1	0	0
OPC	op.	1	0	1	0	X
LD	+	1	0	2	0	X
ST	+	0	0	X	1	1

Jump instruction JMP (before)

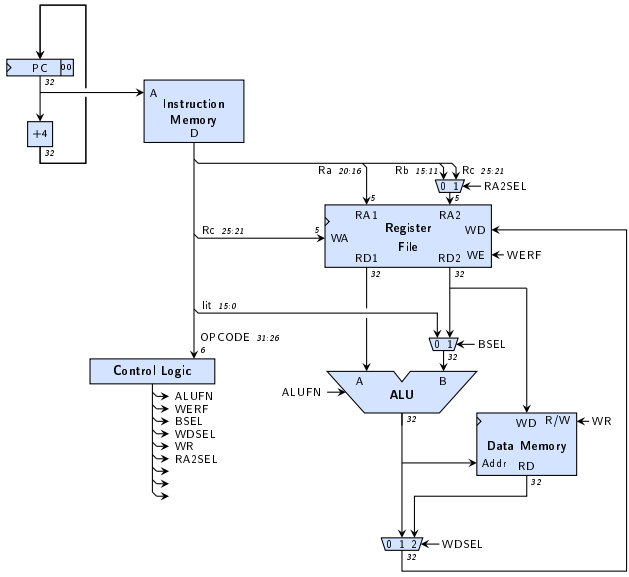
JMP(Ra, Rc)
PC ⟵ PC + 4
Reg[Rc] ⟵ PC
PC ⟵ Reg[Ra] & 0xFFFFFFFC



Jump instruction JMP (before)

JMP(Ra, Rc)
PC ⟵ PC + 4
Reg[Rc] ⟵ PC
PC ⟵ Reg[Ra] & 0xFFFFFFFC

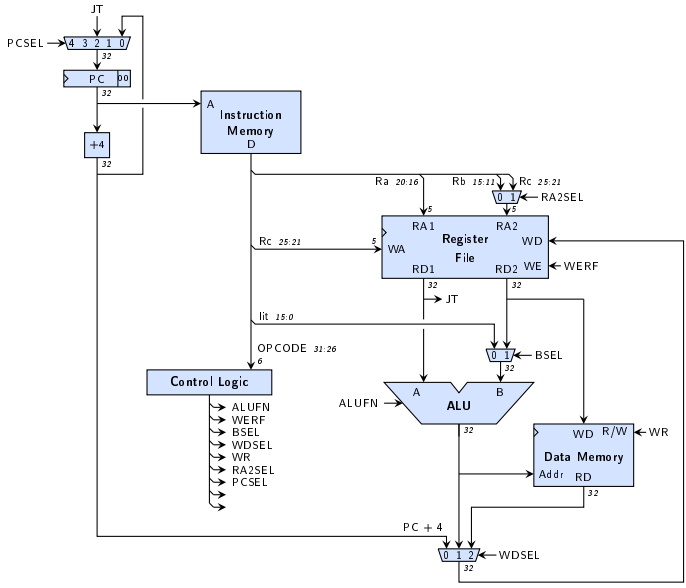
- The ALU and memory are ignored



Jump instruction JMP (after)

JMP(Ra, Rc)
PC ⟵ PC + 4
Reg[Rc] ⟵ PC
PC ⟵ Reg[Ra] & 0xFFFFFFFC

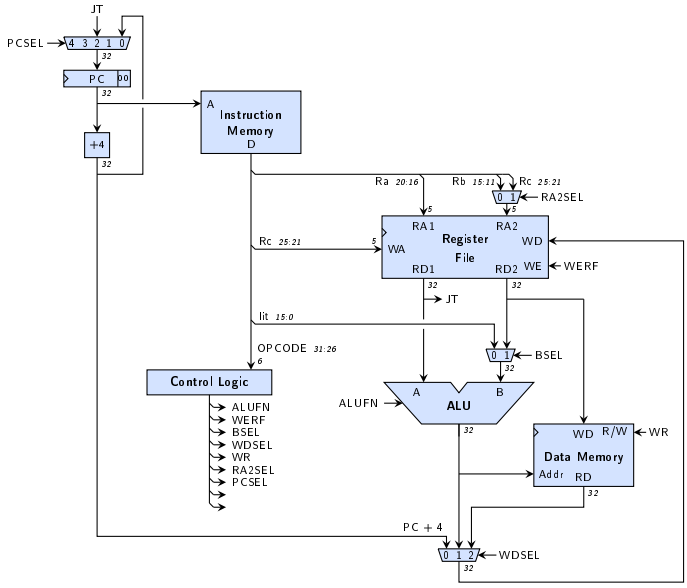
- The ALU and memory are ignored
- The first output of the register file given as value to store in PC at the next clock trigger



Jump instruction JMP (after)

JMP(Ra, Rc)
PC ⟵ PC + 4
Reg[Rc] ⟵ PC
PC ⟵ Reg[Ra] & 0xFFFFFFFC

- The ALU and memory are ignored
- The first output of the register file given as value to store in PC at the next clock trigger
- The result of PC + 4 is given as data to register file for writing at the next clock trigger



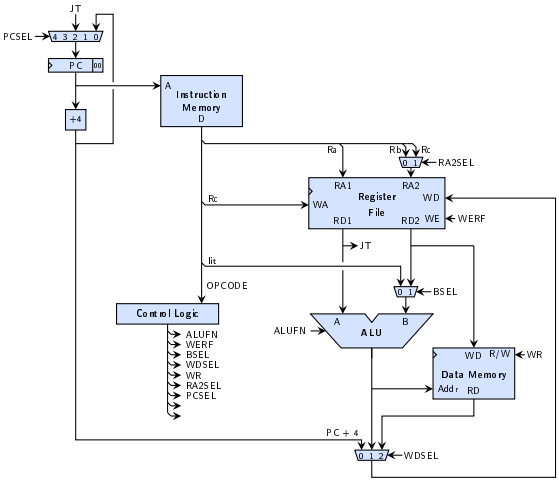
Jump instruction JMP (description)

- The control logic unit sets PCSEL to 2, so that the next value in PC will be JT (that is, the content of register Ra, as output on RD1)

WDSEL[2] selects either PC + 4 (0), the ALU result (1), or the read data from the memory (2) as write data for the register file

PCSEL[2] selects the next PC value: either PC + 4 (0) or the first output (JT) of the register file (2)
Option (1) will be used later

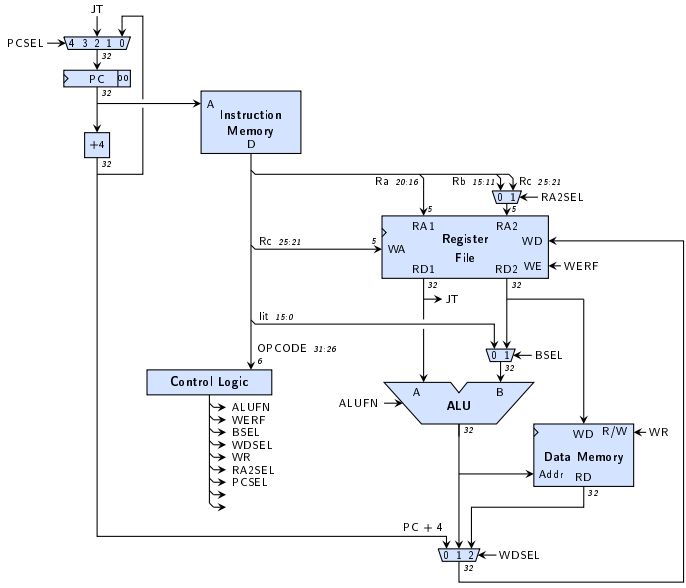
Jump instruction JMP (control logic)



	ALUFN	WERF	BSEL	WDSEL	WR	RA2SEL	PCSEL
OP	op.	1	1	1	0	0	0
OPC	op.	1	0	1	0	X	0
LD	+	1	0	2	0	X	0
ST	+	0	0	X	1	1	0
JMP	X	1	X	0	0	X	2

Branching instructions BEQ and BNE (before)

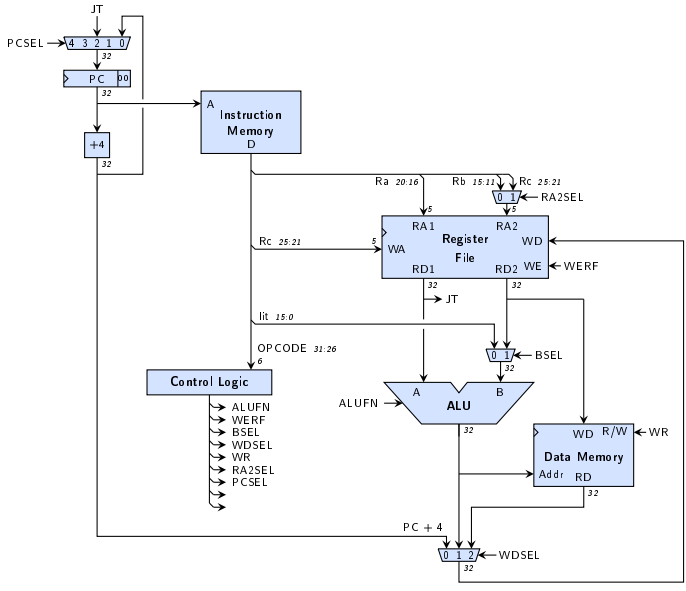
BEQ(Ra, label, Rc)
PC ⟵ PC + 4
Reg[Rc] ⟵ PC
if Reg[Ra] = 0 **then**
PC ⟵ PC + 4 × SXT(literal)



Branching instructions BEQ and BNE (before)

BEQ(Ra, label, Rc)
PC ⟵ PC + 4
Reg[Rc] ⟵ PC
if Reg[Ra] = 0 **then**
PC ⟵ PC + 4 × SXT(literal)

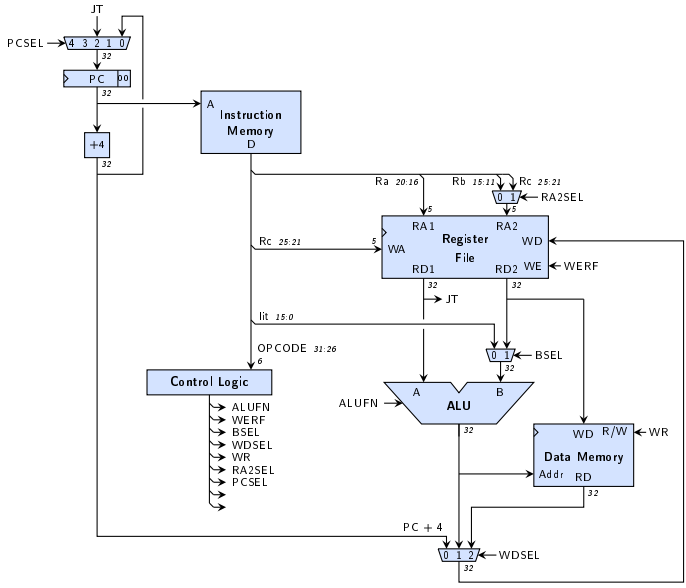
- On register file/ALU/memory side, this is similar to JMP



Branching instructions BEQ and BNE (before)

BEQ(Ra, label, Rc)
PC ⟵ PC + 4
Reg[Rc] ⟵ PC
if Reg[Ra] = 0 **then**
PC ⟵ PC + 4 × SXT(literal)

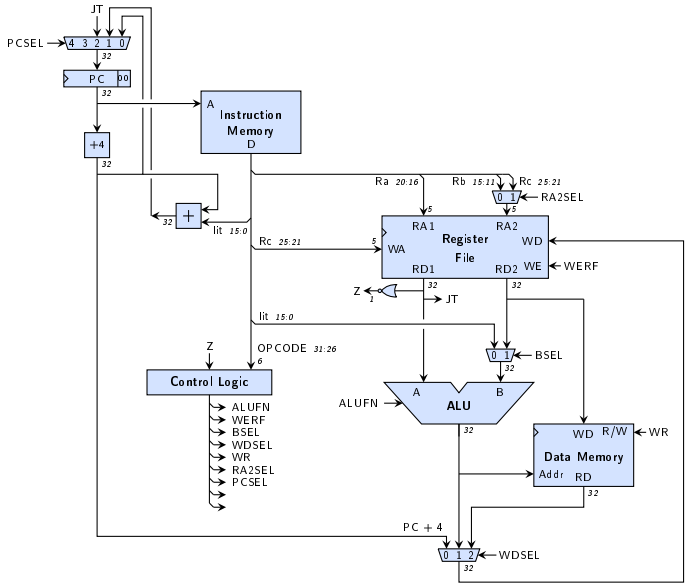
- On register file/ALU/memory side, this is similar to JMP
- PC + 4 given as data to register file for writing at the next clock trigger



Branching instructions BEQ and BNE (after)

BEQ(Ra, label, Rc)
PC ⟵ PC + 4
Reg[Rc] ⟵ PC
if Reg[Ra] = 0 **then**
PC ⟵ PC + 4 × SXT(literal)

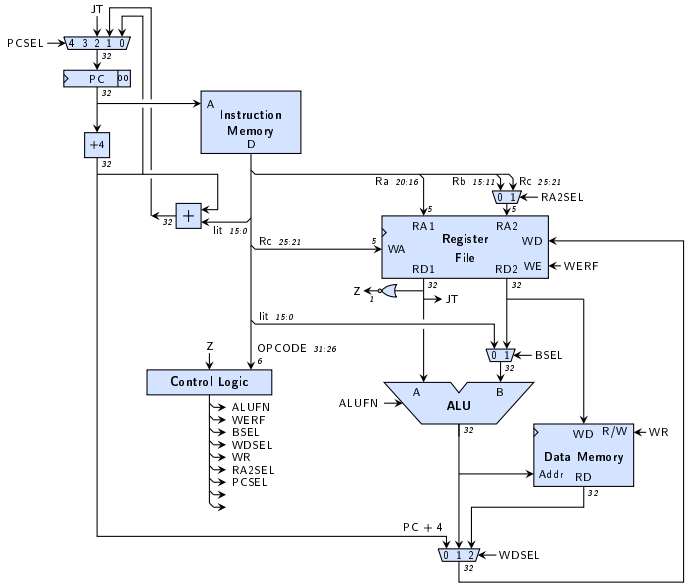
- On register file/ALU/memory side, this is similar to JMP
- PC + 4 given as data to register file for writing at the next clock trigger
- PC ⟵ PC + 4 × SXT(literal) is computed by a simple additional circuit



Branching instructions BEQ and BNE (after)

BEQ(Ra, label, Rc)
PC ⟵ PC + 4
Reg[Rc] ⟵ PC
if Reg[Ra] = 0 **then**
PC ⟵ PC + 4 × SXT(literal)

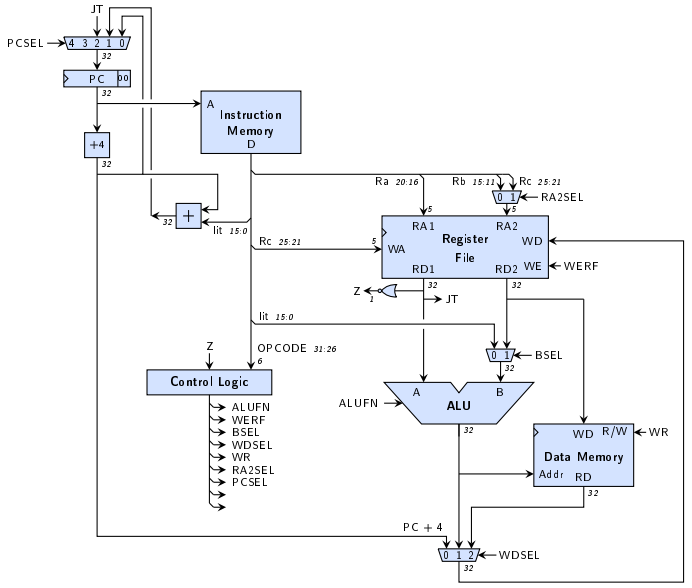
- On register file/ALU/memory side, this is similar to JMP
- PC + 4 given as data to register file for writing at the next clock trigger
- PC ⟵ PC + 4 × SXT(literal) is computed by a simple additional circuit
- PCSEL here depends on Reg[Ra]



Branching instructions BEQ and BNE (after)

BEQ(Ra, label, Rc)
PC ⟵ PC + 4
Reg[Rc] ⟵ PC
if Reg[Ra] = 0 **then**
PC ⟵ PC + 4 × SXT(literal)

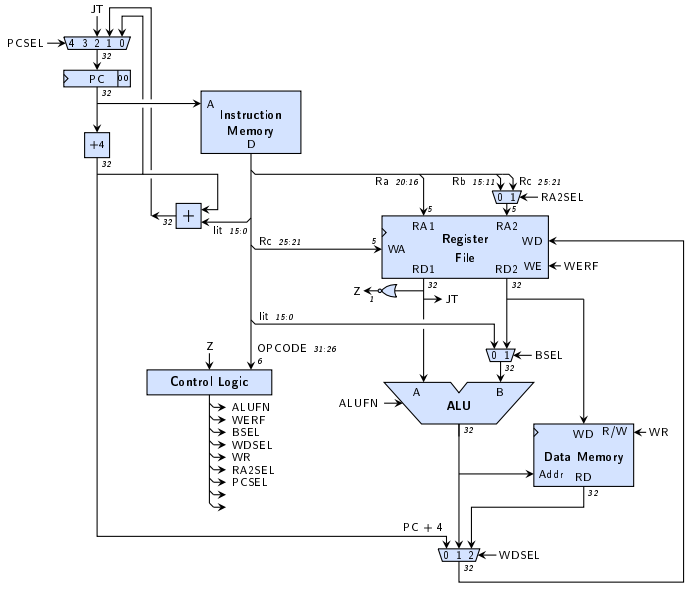
- On register file/ALU/memory side, this is similar to JMP
- PC + 4 given as data to register file for writing at the next clock trigger
- PC ⟵ PC + 4 × SXT(literal) is computed by a simple additional circuit
- PCSEL here depends on Reg[Ra]
- The control logic selects the right PCSEL, depending on Z



Branching instructions BEQ and BNE (after)

BEQ(Ra, label, Rc)
PC ⟵ PC + 4
Reg[Rc] ⟵ PC
if Reg[Ra] = 0 **then**
PC ⟵ PC + 4 × SXT(literal)

- On register file/ALU/memory side, this is similar to JMP
- PC + 4 given as data to register file for writing at the next clock trigger
- PC ⟵ PC + 4 × SXT(literal) is computed by a simple additional circuit
- PCSEL here depends on Reg[Ra]
- The control logic selects the right PCSEL, depending on Z
- Z is simply computed by ORing all bits of Reg[Ra], and taking the negation

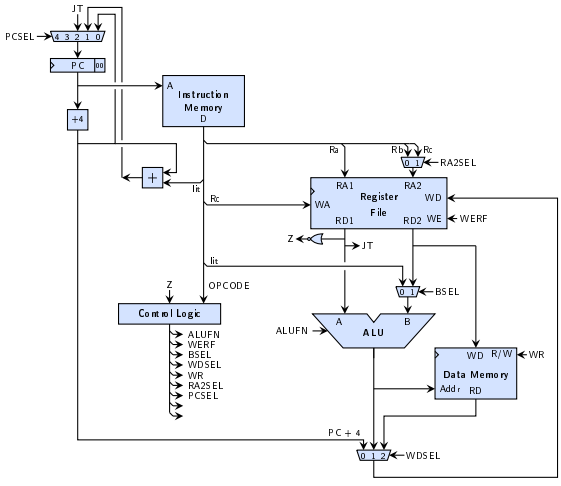


Branching instructions BEQ and BNE (description)

- An additional link between the instruction and the program counter
- depends of the value of Z, itself computed from the first output of the register file

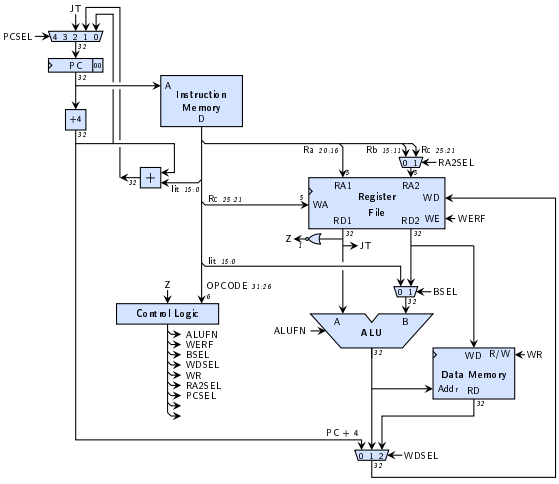
  $PCSEL^2$ selects the next PC value: either PC + 4 (0), the sum 4 * SXT(literal) with PC + 4 (1) or the first output (JT) of the register file (2)

Branching instructions BEQ and BNE (control logic)



	ALUFN	WERF	BSEL	WDSEL	WR	RA2SEL	PCSEL
OP	op.	1	1	1	0	0	0
OPC	op.	1	0	1	0	X	0
LD	+	1	0	2	0	X	0
ST	+	0	0	X	1	1	0
JMP	X	1	X	0	0	X	2
BEQ	X	1	X	0	0	X	Z
BNE	X	1	X	0	0	X	¬Z

The full β machine



	ALUFN	WERF	BSEL	WDSEL	WR	RA2SEL	PCSEL
OP	op.	1	1	1	0	0	0
OPC	op.	1	0	1	0	X	0
LD	+	1	0	2	0	X	0
ST	+	0	0	X	1	1	0
JMP	X	1	X	0	0	X	2
BEQ	X	1	X	0	0	X	Z
BNE	X	1	X	0	0	X	¬Z

Control logic

ALUFN[4]	ALU operation ($+$, $-$, $*$, $/$, $\wedge$, $\vee$, $\otimes$, $=$, $<$, $\leq$, $\ll$, $\gg$, $\gg'$)
WERF[1]	enable (1) or not (0) writing in a register
BSEL[1]	select the constant embedded in instruction (0) or a register (1) as the B argument for the ALU
WDSEL[2]	select PC + 4 (0), the ALU result (1), or the data read from the memory (2) as data to be written in the register file
WR[1]	enable (1) or not (0) writing in the memory
RA2SEL[1]	select instruction bits 15:11 or 25:21 as second read address for the register file
PCSEL[2]	selects the next PC value: either PC + 4 (0), the sum 4 * SXT(literal) with PC + 4 (1) or the first output (JT) of the register file (2)

The full β machine (In Logisim)