Kabeta Processor Design

|  |  |
| --- | --- |
| Date: | April 25, 2018 |
| Version: | 1.1B |
| Author: | Katherine White |
| Reviewer: | (N/A) |

# Introduction

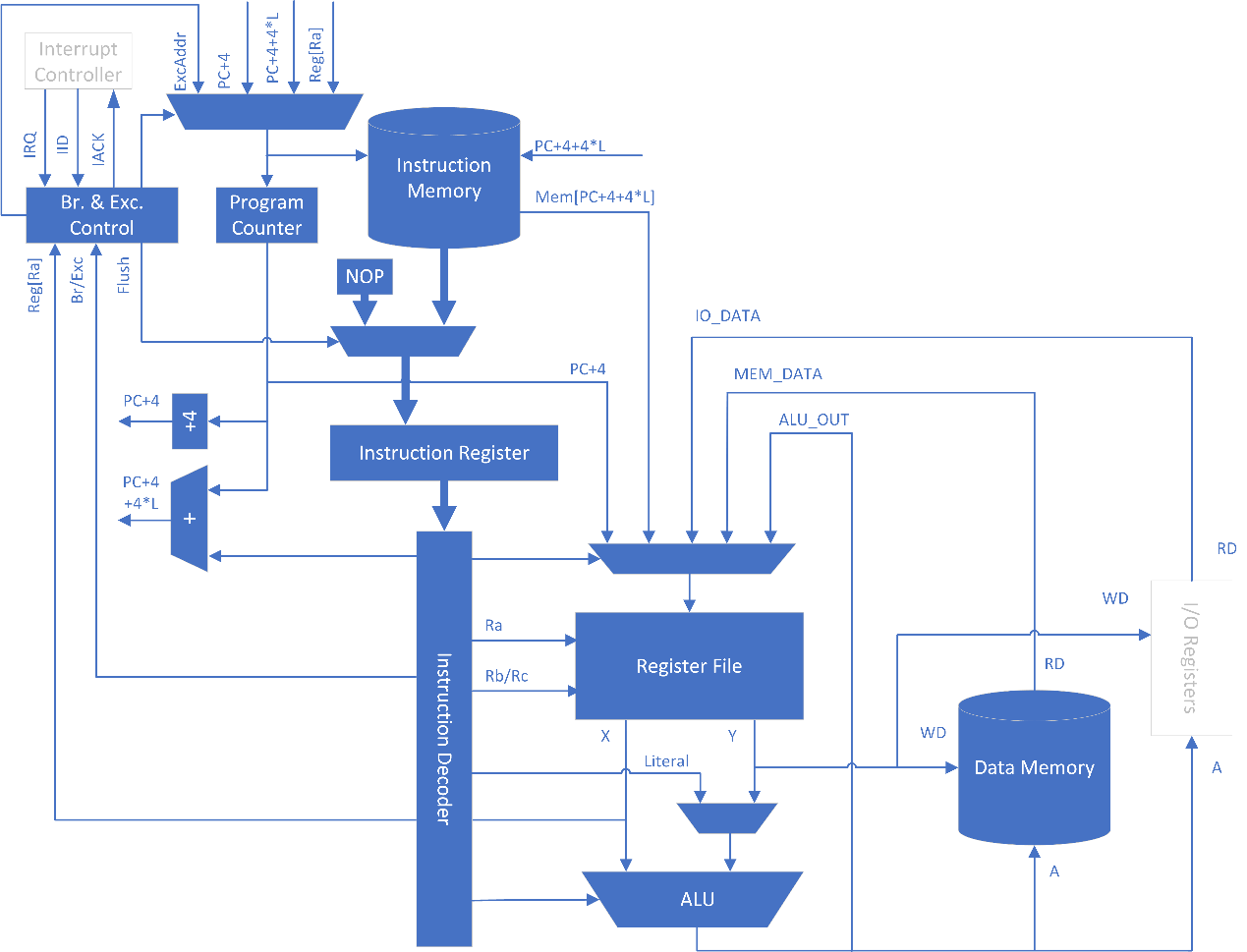
## Description

Kabeta is a RISC processor based on the β Processor of MIT. Its main features and limitations include:

* Typical 5-stage Pipeline with Bypass
* Supervisor and User Modes
* Separate Instruction, Data and I/O Address Spaces
* Synchronous Single-cycle Access on-chip Instruction and Data RAMs

(for the sake of implementation in FPGA)

## Components



Kabeta mainly consists of Register File, Arithmetic and Logic Unit, Instruction Registers and Decoders, Branch and Exception Control, Address Adders, Program Counter, on-chip Data Memory and Instruction Memory.

## Block Diagram

[Open Detailed Block Diagram (Picture](Design%20Diagrams%20-%20Detailed%20Block%20Diagram.png))

[Go to Detailed Block Diagram](#DetailedBlockDiagram)

**NOTES:**

* ALU in the diagram should contain an output register.
* Read-while-write behavior of Register File should be write-through, i.e. the output data should be the data to be written.

## References

* [MIT β Processor Specification](MIT6_004s09_lab_beta_doc.pdf)
* [MIT β Processor Summary](MIT6_004s09_lab_beta_summary.pdf)
* [Lecture Notes](https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-004-computation-structures-spring-2009/lecture-notes/) of [MIT 6.004 Computation Structures](https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-004-computation-structures-spring-2009/) ([L14](MIT6_004s09_lec14.pdf), [L22](MIT6_004s09_lec22.pdf), [L23](MIT6_004s09_lec23.pdf))

# Instruction Extension

## System Service – SVC (1C)

|  |  |  |  |
| --- | --- | --- | --- |
| 31 26 | 25 21 | 20 16 | 15 0 |
| **01**1100 | 00000 | 00000 | SVC\_ID |

**NOTE:** This instruction will cause System Service Trap.

## Input/Output Read – IOR (08)

|  |  |  |  |
| --- | --- | --- | --- |
| 31 26 | 25 21 | 20 16 | 15 0 |
| **00**1000 | Rc | Ra | Offset (two’s complement) |

Reg[Rc] ← IO[Reg[Ra] + Sext(Offset)]

**NOTE:** This is a privileged instruction.

## Input/Output Write – IOW (09)

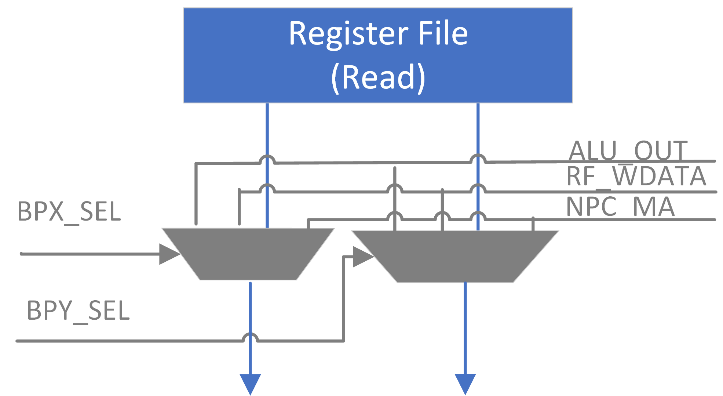
|  |  |  |  |
| --- | --- | --- | --- |
| 31 26 | 25 21 | 20 16 | 15 0 |
| **00**1001 | Rc | Ra | Offset (two’s complement) |

IO[Reg[Ra] + Sext(Offset)] ← Reg[Rc]

**NOTE:** This is a privileged instruction.

# Bypass

## Bypass Paths



## Control Signals

* Port X

ALU\_OUT\_SELX = (IR\_EX.Opcode in {OP, OPC, LD, ST, JMP, B, IOR, IOW})

&& (IR\_EX.Ra != 31)

&& (IR\_EX.Ra == IR\_MA.Rc) && (IR\_MA.Opcode in {OP, OPC})

NPC\_MA\_SELX = (IR\_EX.Opcode in {OP, OPC, LD, ST, JMP, B, IOR, IOW})

&& (IR\_EX.Ra != 31)

&& (IR\_EX.Ra == IR\_MA.Rc) && (IR\_MA.Opcode in {JMP, B})

RF\_WDATA\_SELX = (IR\_EX.Opcode in {OP, OPC, LD, ST, JMP, B, IOR, IOW})

&& (IR\_EX.Ra != 31)

&& (!ALU\_OUT\_SELX && !NPC\_MA\_SELX)

&& (IR\_EX.Ra == IR\_WB.Rc)

&& (IR\_WB.Opcode in {OP, OPC, LD, LDR, JMP, B, IOR})

* Port Y

ALU\_OUT\_SELY = **(**(IR\_EX.Opcode in {OP}) && (IR\_EX.Rb != 31)

&& (IR\_EX.Rb == IR\_MA.Rc) && (IR\_MA.OpCode in {OP, OPC})**)**

|| **(**(IR\_EX.Opcode in {ST, IOW}) && (IR\_EX.Rc != 31)

&& (IR\_EX.Rc == IR\_MA.Rc) && (IR\_MA.OpCode in {OP, OPC})**)**

NPC\_MA\_SELY = **(**(IR\_EX.Opcode in {OP}) && (IR\_EX.Rb != 31)

&& (IR\_EX.Rb == IR\_MA.Rc)

&& (IR\_MA.Opcode in {JMP, B})**)**

|| **(**(IR\_EX.Opcode in {ST, IOW}) && (IR\_EX.Rc != 31)

&& (IR\_EX.Rc == IR\_MA.Rc)

&& (IR\_MA.Opcode in {JMP, B})**)**

RF\_WDATA\_SELY = **(**(IR\_EX.Opcode in {OP}) && (IR\_EX.Rb != 31)

&& (!ALU\_OUT\_SELY && !NPC\_MA\_SELY)

&& (IR\_EX.Rb == IR\_WB.Rc)

&& (IR\_WB.Opcode in {OP, OPC, LD, LDR, JMP, B, IOR})**)**

|| **(**(IR\_EX.Opcode in {ST, IOW}) && (IR\_EX.Rc != 31)

&& (!ALU\_OUT\_SELY && !NPC\_MA\_SELY)

&& (IR\_EX.Rc == IR\_WB.Rc)

&& (IR\_WB.Opcode in {OP, OPC, LD, LDR, JMP, B, IOR})**)**

# Pipeline Stall

Stall the pipeline when one of the source registers of the instruction at RR-Stage coincides with the target register of the load/read instruction at EX-Stage.

## Control Signals

Stall = **(**(IR\_RR.Opcode in {OP, OPC, LD, ST, JMP, B, IOR, IOW}) // Read Ra

&& (IR\_RR.Ra != 31)

&& (IR\_RR.Ra == IR\_EX.Rc) && (IR\_EX.Opcode in {LD, LDR, IOR})**)**

|| **(**(IR\_RR.Opcode in {OP}) // Read Rb

&& (IR\_RR.Rb != 31)

(IR\_RR.Rb == IR\_EX.Rc) && (IR\_EX.OpCode in {LD, LDR, IOR})**)**

|| **(**(IR\_RR.Opcode in {ST, IOW}) // Read Rc

&& (IR\_RR.Rc != 31)

&& (IR\_RR.Rc == IR\_EX.Rc) && (IR\_EX.Opcode in {LD, LDR, IOR})**)**

NOTE: Stall the pipeline as early as possible to disable less components.

## Implementation

Inject NOP instruction into EX-stage and disable Register File read, PC\_RR, IR\_RR, PC\_IF and Instruction Memory.

**NOTE:** This implementation is a bit different from MIT β Processor.

## Mitigation

When writing programs, put an instruction independent of the data following the load/read instruction to eliminate pipeline stall.

# Exception

## Reference

Refer to Section 6. Extensions for Exception Handling in [MIT β Processor Specification](file:///F:\Workspace\Kabeta\doc\MIT6_004s09_lab_beta_doc.pdf)

## Supported Exceptions

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Code** | **Type** | **Source** | **Priority** | **Exc. Vector** |
| Reset | 000 | Reset | RST Pin | 0 (highest) | 8000\_0000 |
| System Service | 001 | Trap | RR-Stage | 3 | 8000\_0004 |
| Illegal Instruction | 010 | Fault | RR-Stage | 8000\_0008 |
| Invalid Operation | 011 | Fault | EX-Stage | 2 | 8000\_000C |
| Invalid D-Address | 100 | Fault | MA-Stage | 1 | 8000\_0010 |
| Invalid I-Address | 101 | Fault | IF-Stage | 4 | 8000\_0014 |
| Interrupt 0 | 110 | Interrupt | IRQ Pin | 5 (lowest) | 8000\_0018 |
| Interrupt 1 | 111 | Interrupt | IRQ Pin | 8000\_001C |

**NOTES:**

1) The MSBs of exception vectors indicate that the exception handlers will executed in the Supervisor Mode.

2) If instructions at multiple stages have caused exceptions simultaneously, only the exception in the highest priority should be processed.

3) Pay attention to the suppression of exceptions by the branch instruction in EX-Stage, refer to Section 6.2.

## Exception Conditions

|  |  |
| --- | --- |
| **Name** | **Condition** |
| Reset | External reset signal (active LOW) |
| System Service | SVC instruction |
| Illegal Instruction | Undefined opcodes (6’b010\_xxx, 6’b011\_010) in both modes, privileged opcodes (6’b00x\_xxx) in User Mode, or undefined privileged opcodes in Supervisor Mode. |
| Invalid Operation | Undefined Operate Class Opcodes, MUL/MULC, or DIV/DIVC |
| Invalid D-Address | Out of data memory address range |
| Invalid I-Address | Out of instruction memory address range |
| Interrupt 0/1 | External interrupt signal |

## Interrupt Request and Acknowledge

* External interrupts are level triggered.
* Interrupt number is read from IID pin.
* Acknowledge is a pulse of one clock cycle’s duration.

**NOTE:** The above signals (IRQ, IID and IACK) cross clock domains.

[Open Interrupt Request and Acknowledge Timing Diagram (Picture)](IntInterface.png)

[Go to Interrupt Request and Acknowledge Timing Diagram](#InterruptSyncTiming)

## Implementation

When BNE(R31,0,XP) instruction arrives at WB-Stage, write register XP = PC\_WB, which is the instruction address plus 4.

### Reset Processing

* Reset all Instruction Registers (i.e. load NOPs).
* Reset all Program Counters (i.e. load 32’h0000\_0000 address).
* Set ExcAddr = reset exception vector, and select ExcAddr as next PC value.

**NOTE:** Synchronization of external RST signal is necessary.

### Trap and Fault Processing

When a trap or fault occurs:

* Replace the instruction which has caused the exception with BNE(R31,0,XP) instruction.
* Replace the later instructions in the pipeline with NOP instructions.
* Jump to the exception handler.

**NOTE:** Traps and Faults could be nested, as long as XP is stored onto stack in the exception handlers.

### Interrupt Processing

When an interrupt occurs:

* Wait until Supervisor bit is cleared (i.e. PC\_IF.S == 0).
* Replace the instruction at EX-Stage with BNE(R31,0,XP) instruction.
* Replace the instructions at IF-Stage and RR-Stage with NOP instructions.
* Jump to the exception handler.

**NOTE:** Interrupts could not be nested with other exceptions. (Because PC\_IF.S will be set when the first instruction of the IRQ handler is fetched.)

# Branch

## Branch Delay Slots

There are 2 branch delay slots. The instructions in the branch delay slots will be flushed by asserting *FlushIF* and *FlushRR*, and corresponding PCs will be replaced with PC\_EX (i.e. the address of the branch instruction plus 4) by asserting *ReplicatePC*, if the branch is taken.

## Exception Suppression

When branch instruction is at EX-Stage and branch is taken, the Traps and Faults caused by the instructions in the branch delay slots will be ignored.

## Supervisor Mode

Only the JMP instruction is allowed to clear the Supervisor bit but not set it, and no other instructions may have any effect on it. If JMP tries to switch to Supervisor Mode from User Mode, the S bit will be ignored and no exception will be caused.

# Components Signaling

## Instruction Decoder Signaling

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Signal** | | **OP** | **OPC** | **LD** | **IOR** | **ST** | **IOW** | **JMP** | **BEQ** | **BNE** | **LDR** | **SVC** |
| IF | (N/A) |  |  |  |  |  |  |  |  |  |  |  |
| RR | RegAddrYSel | RB | X | X | X | RC | RC | X | X | X | X | X |
| RegRdEnX | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| RegRdEnY | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| EX | BrCond**[1]** | NV | NV | NV | NV | NV | NV | AL | EQ | NE | NV | NV |
| InstrnMemAddrBufEn | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| ALU\_DataYSel | REG | LIT | LIT | LIT | LIT | LIT | X | X | X | X | X |
| MemDataBufEn | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| ALU\_Op | ALU\_Op | ALU\_Op | ADD | ADD | ADD | ADD | X | X | X | X | X |
| ALU\_En | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| MA | InstrnMemRdEn | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| MemIO\_Sel | X | X | MEM | IO | MEM | IO | X | X | X | X | X |
| MemIO\_Ren | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| MemIO\_Wen | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| ALU\_DataBufEn | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| WB | RegDataWSel**[2]** | ALU | ALU | ALU | ALU | X | X | PC | PC | PC | IM | X |
| RegWrEn | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |

**NOTE:**

1) BrCond:

NV – Do not take

AL – Take unconditionally

EQ – Take if equal

NE – Take if not equal

\*) ‘X’ means Don’t Care.

2) RegDataWSel:

PC – PC+4

ALU – ALU Data Buffer

MEM – Data Memory

IO – IO Registers

IM – Instruction Memory

## Branch and Exception Control Signaling

BrTaken = **(**BrCond == AL**)**

|| **(**(BrCond == EQ) && RaZero**)**

|| **(**(BrCond == NE) && !RaZero**)**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Exc. Req.  && Cond.  Exc. Code  Priority | **RST** | **ExcReqMA** | **ExcReqEX** | **ExcReqRR**  **&& !BrTaken**  **&& !Stall** | **ExcReqIF**  **&& !BrTaken**  **&& !Stall** | **IRQ**  **&& (~PC\_IF.S)** | **BrCond[3]** | | | | **Stall[3]** | **(None Assert)** |
| AL | EQ &&  RaZero | | NE &&  !RaZero |
| **(N/A)** | **ExcCodeMA** | **ExcCodeEX** | **ExcCodeRR** | **ExcCodeIF** | **(N/A)** | **(N/A)** | | | | **(N/A)** |  |
| 0 (Highest) | 1 | 2 | 3 | 4 | 5 | 6 (Lowest) | | | | 6 (Lowest) |  |
| **ExcAddr** | 32’h8000\_0000 | EVT[ExcCodeMA] | EVT[ExcCodeEX] | EVT[ExcCodeRR] | EVT[ExcCodeIF] | EVT[{2’b11,IID}] | X | | | | X | X |
| **PC\_Sel[1]** | EXCA | EXCA | EXCA | EXCA | EXCA | EXCA | REGA | | PCLIT | PCLIT | X | PCNX |
| **FlushIF** | 0 | 1 | 1 | 1 | 1 | 1 | 1 | | | | 0 | 0 |
| **ExcAckIF[2]** | 0 | 0 | 0 | 0 | 1 | 0 | 0 | | | | 0 | 0 |
| **FlushRR** | 0 | 1 | 1 | 1 | 0 | 1 | 1 | | | | 1 | 0 |
| **ExcAckRR[2]** | 0 | 0 | 0 | 1 | 0 | 0 | 0 | | | | 0 | 0 |
| **FlushEX** | 0 | 1 | 1 | 0 | 0 | 1 | 0 | | | | 0 | 0 |
| **ExcAckEX[2]** | 0 | 0 | 1 | 0 | 0 | 1 | 0 | | | | 0 | 0 |
| **FlushMA** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | | | | 0 | 0 |
| **ExcAckMA[2]** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | | | | 0 | 0 |
| **ReplicatePC** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | | | | 0 | 0 |

**NOTE:**

1) PC\_Sel

EXCA – Exception Address

PCNX – PC+4

PCLIT – PC+4+4\*Sext(Literal)

REGA – Register Ra

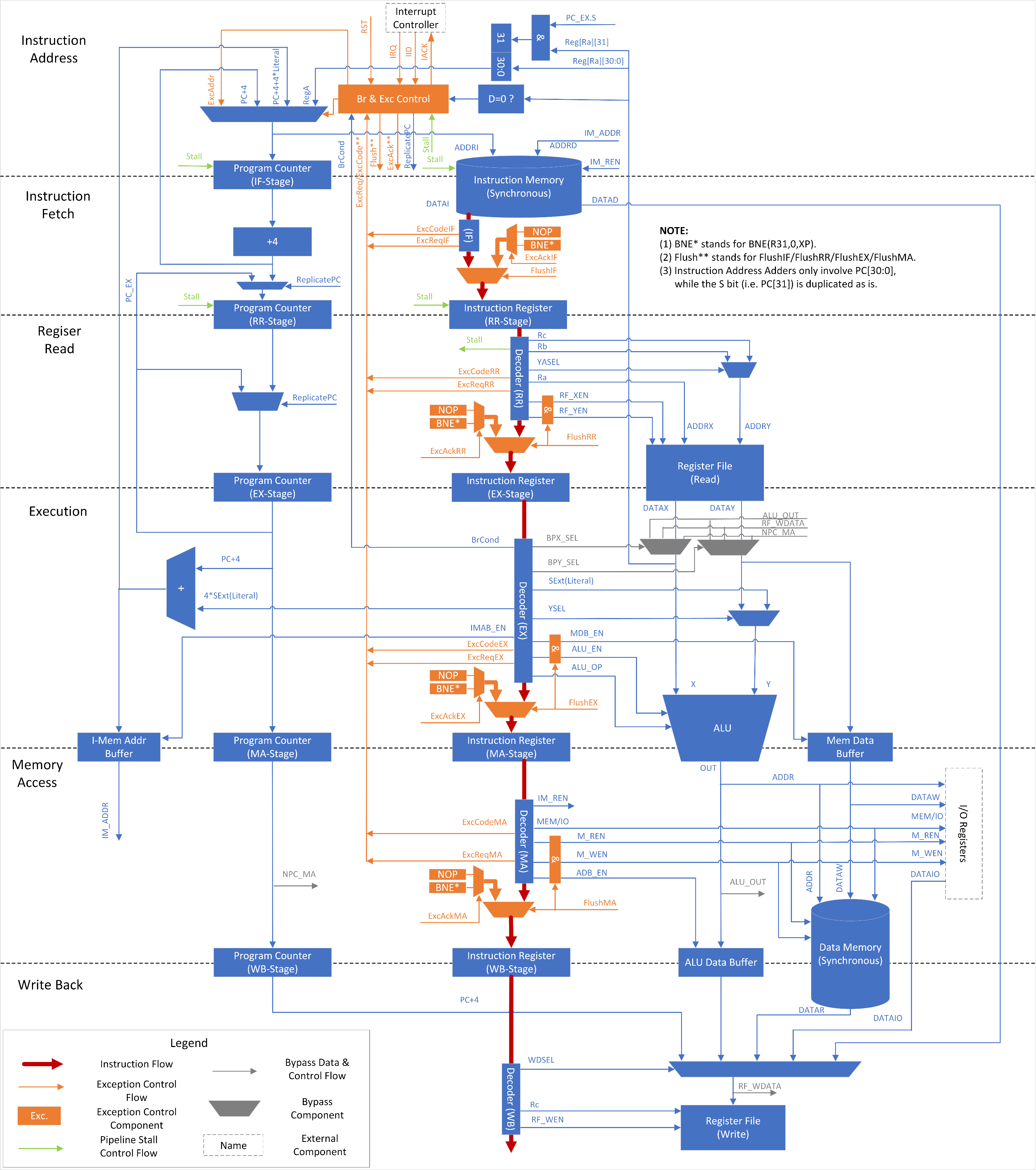
2) ExcAck\*

0 – Select NOP

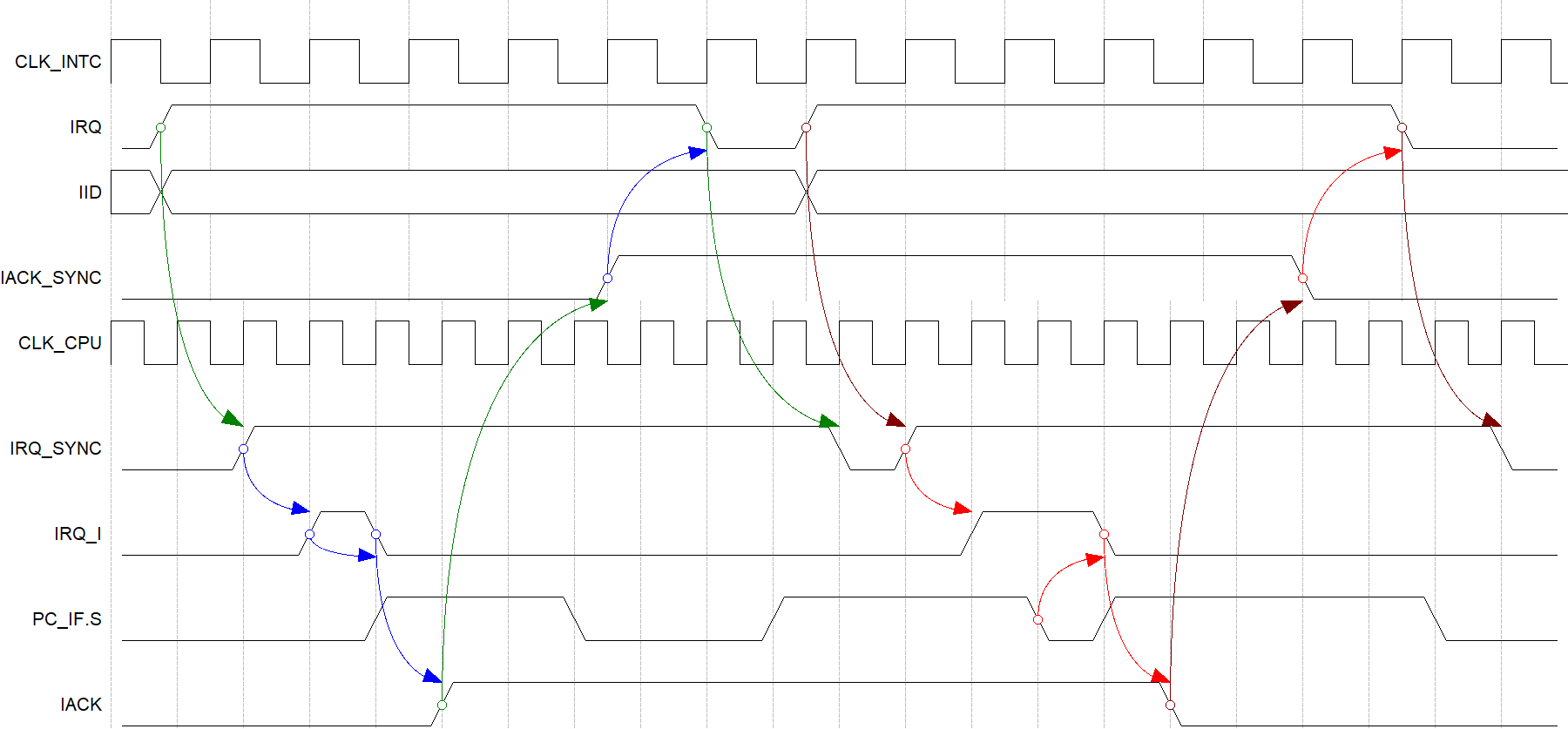
1 – Select BNE

\*) ‘X’ means Don’t Care.

3) BrCond and Stall will not assert simultaneously.



Detailed Block Diagram



Interrupt Request and Acknowledge Timing Diagram

# Appendix A: Document Version History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Date** | **Editor** | **Reviewer** | **Comment** |
| 1.0 | 4/11/2018 | K. White | (N/A) | Initial version. |
| 1.0B | 4/12/2018 | K. White | Dao Cat | 1) Elaborate top level block diagram.  2) Replace NOP with BNE(R31,0,XP).  3) Put detailed block diagram into this document. |
| 1.1 | 4/15/2018 | K. White | (N/A) | Add decoder signaling table. |
| 1.1A | 4/17/2018 | K. White | (N/A) | Add branch and exception signaling table. |
| 1.1B | 4/25/2018 | K. White | (N/A) | Add interrupt request and acknowledge timing diagram. |