Kabeta Processor Design

|  |  |
| --- | --- |
| Date: | April 9, 2018 |
| Version: | Preliminary |
| Author: | Kathy White |
| Reviewer: | (N/A) |

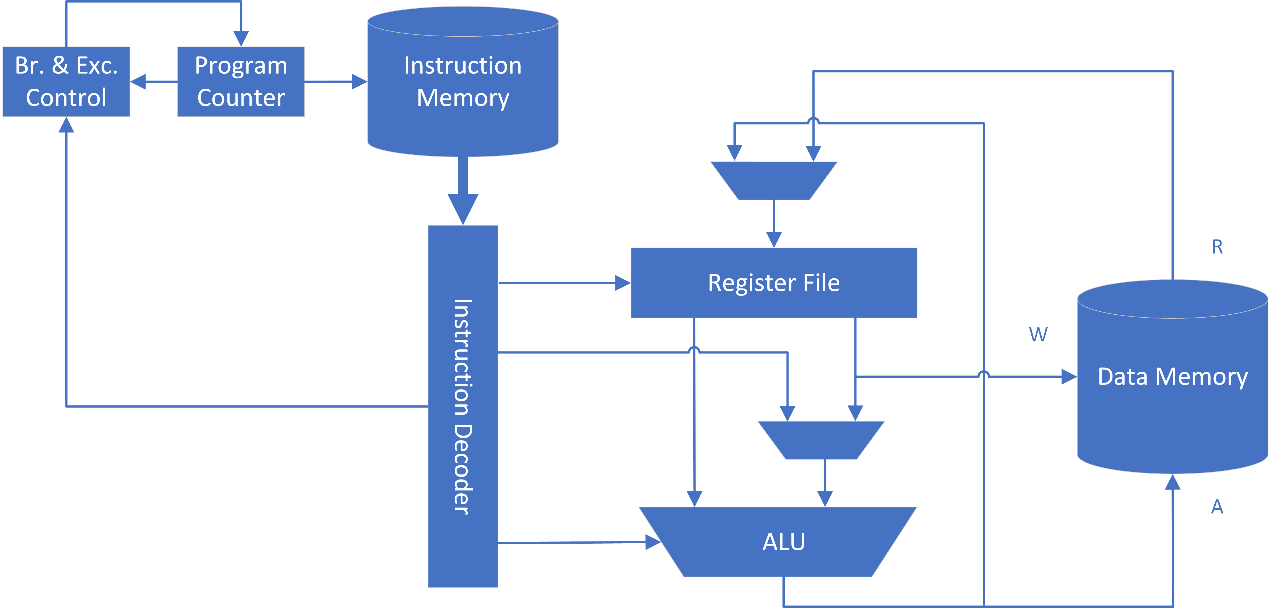
# Introduction

## 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 and Data Space
* Single-cycle Synchronous on-chip RAM

## Components



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

## Block Diagram

[Click Here to Open Detailed Block Diagram](Design%20Diagrams%20-%20Detailed%20Block%20Diagram.png)

**IMPORTANT NOTE:**

* 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

## No Operation Instruction -- NOP (1A)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 31 26 | 25 21 | 20 16 | 15 1 | 0 |
| **01**1010 | 00000 | 00000 | 00000000000 | E |

E: Exception Status,

0 – Normal

1 – Exceptional

**NOTE:** NOP instruction **does not** cause exceptions.

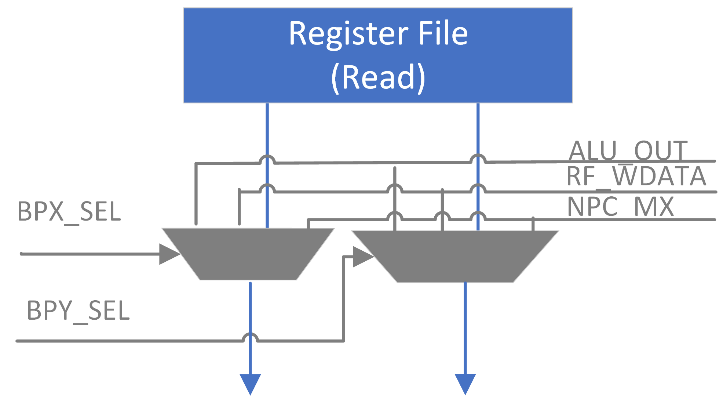
## System Service Instruction -- SVC (1C)

|  |  |  |  |
| --- | --- | --- | --- |
| 31 26 | 25 21 | 20 16 | 15 0 |
| **01**1100 | 00000 | 00000 | 000000000000 |

This instruction will cause System Service Exception.

# Bypass

## Bypass Paths



## Control Signals

BPX\_SEL: (IR\_EX.Opcode in {OP, OPC, LD, ST, JMP, B\*}) // Instrns Read Ra

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

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

NPC\_MX\_SELX : (IR\_EX.Ra == IR\_MA.Rc) && (IR\_MA.Opcode in {JMP, B\*})

RF\_WDATA\_SELX : (!ALU\_OUT\_SELX && !NPC\_MX\_SELX) && (IR\_EX.Ra == IR\_WB.Rc) && (IR\_WB.Opcode in {OP, OPC, LD, LDR, JMP, B\*})

BPY\_SEL: (IR\_EX.Opcode in {OP}) // Instrns Read Rb

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

ALU\_OUT\_SELY : (IR\_EX.Rb == IR\_MA.Rc) && (IR\_MA.OpCode in {OP, OPC})

NPC\_MX\_SELY : (IR\_EX.Rb == IR\_MA.Rc) && (IR\_MA.Opcode in {JMP, B\*})

RF\_WDATA\_SELY : (!ALU\_OUT\_SELY && !NPC\_MX\_SELY) && (IR\_EX.Rb == IR\_WB.Rc) && (IR\_WB.Opcode in {OP, OPC, LD, LDR, JMP, B\*})

# 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 instruction at EX-Stage.

## Control Signals

STALL <= ((IR\_RR.Opcode in {OP, OPC, LD, ST, JMP, B\*}) // Instrns Read Ra

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

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

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

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

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

|| ((IR\_RR.Opcode in {ST}) // Instrns Read Rc

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

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

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

## Implementation

Inject a Normal 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 with MIT β Processor.

# 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** | **Exc. Code** | **Type** | **Source** | **Exc. Vector** |
| Reset | 000 | Interrupt | RST Pin | 8000\_0000 |
| System Service | 001 | Trap | RR-Stage | 8000\_0004 |
| Illegal Instruction | 010 | Fault | RR-Stage | 8000\_0008 |
| Invalid I-Address | 011 | Fault | IF-Stage | 8000\_000C |
| Invalid D-Address | 100 | Fault | MA-Stage | 8000\_0010 |
| Invalid Operation | 101 | Fault | EX-Stage | 8000\_0014 |
| External Interrupt | 110 | Interrupt | IRQ Pin | 8000\_0018 |
| (Reserved) | 111 | (N/A) | (N/A) | 8000\_001C |

**NOTE:** The MSBs of exception vectors indicate that the exception handlers will executed in the Supervisor Mode.

## Implementation

When an Exceptional NOP 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 Normal 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 an Exceptional NOP instruction (by asserting ExcIF/ExcRR/ExcEx/ExcMA).
* Replace the later instructions in the pipeline with Normal NOP instructions (i.e. flush the pipeline).
* Jump to the exception handler.

### Interrupt Processing

When an interrupt occurs:

* Replace the instruction at IF-Stage with an Exceptional NOP instruction (by asserting ExcIF).
* Jump to the exception handler.

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

### Traps/Faults in Supervisor Mode

In Supervisor Mode, traps and faults will put the processor into an undefined state.

# Branch

Tips: branch slots = 2 instructions, no slots cancellation if branch.