COMP3211 Assignment

z5210146 Chung Lai Lee

z5270589 YiJie Zhao

Piplined Processor from Lab3

- Two write back ports
  - Two registers that go along the pipeline
- Comparison of BNE during ID stage (early detection, less penalty for wrong prediction)
  - Add a not equal comparison block in ID stage

Forwarding, Hazards and Static Prediction

- Forwarding
  - Forwarding unit to EX stage (Forward from MEM and WB stage)
  - Forwarding unit to ID stage for BNE (same module as the one in EX stage)
- Hazard (Load Use Hazard)
  - Hazard Detection Unit controls stalling
    - Gets input from controls of both EX stage and MEM stage
- Branch Prediction
  - Static Prediction (default no branch)
  - Flush when incorrect prediction

Considerations for BNE

- Forwarding is same as others
- Additional requirements to prevent Load Use Hazard (stalling)
  - BNE does compare in ID stage, it needs data immediately in ID stage
  - Unlike other instructions, when BNE is in ID stage, it needs to check data dependency in both EX stage and MEM stage

IF	ID	EX	MEM	WB
	ADD		LW	

IF	ID	EX	MEM	WB
		ADD		LW

IF	ID	EX	MEM	WB
	BNE		LW	

IF	ID	EX	MEM	WB
	BNE			LW

Why Two Forwarding Unit?

- Only one forwarding unit in EX stage
  - BNE can only be determined at EX stage and need to flush two stages

- Only one forwarding unit in ID stage
  - Stall even for Non BNE instruction RAW dependency

IF	ID	EX	MEM	WB
	ADD	ADD		

IF	ID	EX	MEM	WB
	ADD		ADD	

Having two forwarding unit allows early branch detection and minimize stalling

Structures modified from original processor

instruction size: 16bit -> 32bit

opcode	operand1	operand2	operand3
8 bit	8 bit	8 bit	8bit

register file: 16 registers -> 32 registers

address space: 16 -> 256

data memory now has 3 sub memory components

- pattern characters (PA)
- pattern length (PL)
- pattern occurrence (PO)

instruction and data memory have components that does address translation

And all relevant data paths

ISA

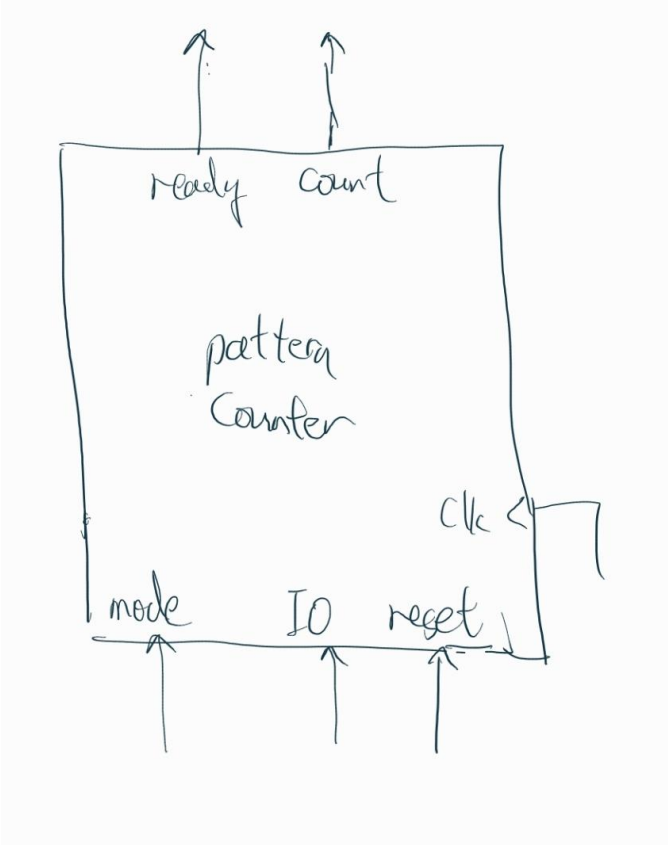
--	BNE R1 R2 imm	if (R1 != R2) branch to PC + 1 + imm
--	ADD R1 R2 R3	R3 <- R1 + R2
--	SOW R1 R2 imm	POA[R1+imm] <- R2
--	LOW R1 R2 imm	R2 <- POA[R1+imm]
--	STB R1 X X	PNReg <- R1
--	LPA R1 R2 R3	if (R2 < R1) {R3 <- PA[offset + R2] and R2++} else {R3 <- -1}
--	LLA R1 R2 R3	if (R2 < R1) {R3 <- LA[R2] and R2++} else {R3 <- -1}
--	LOA R1 R2 R3	if (R2 < R1) {R3 <- OA[R2] and R2++} else {R3 <- -1}
--	MVI R1 X R2	R2 <- reg_file[R1]
--	END X X X	signal end of function (stalls pipeline and signals ready when the instruction reaches mem stage)
--	JP X X imm	branch to PC + 1 + imm

Overview

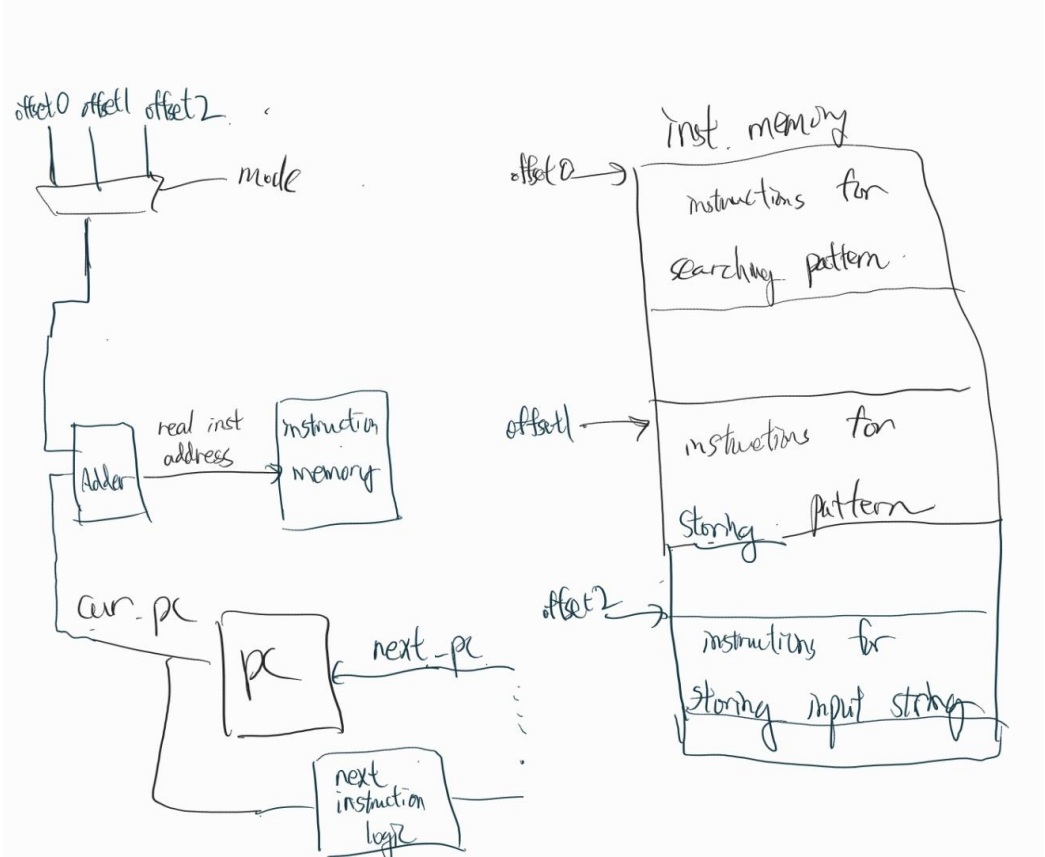
3 function:

- stores pattern from IO
- stores input string from IO
- compare

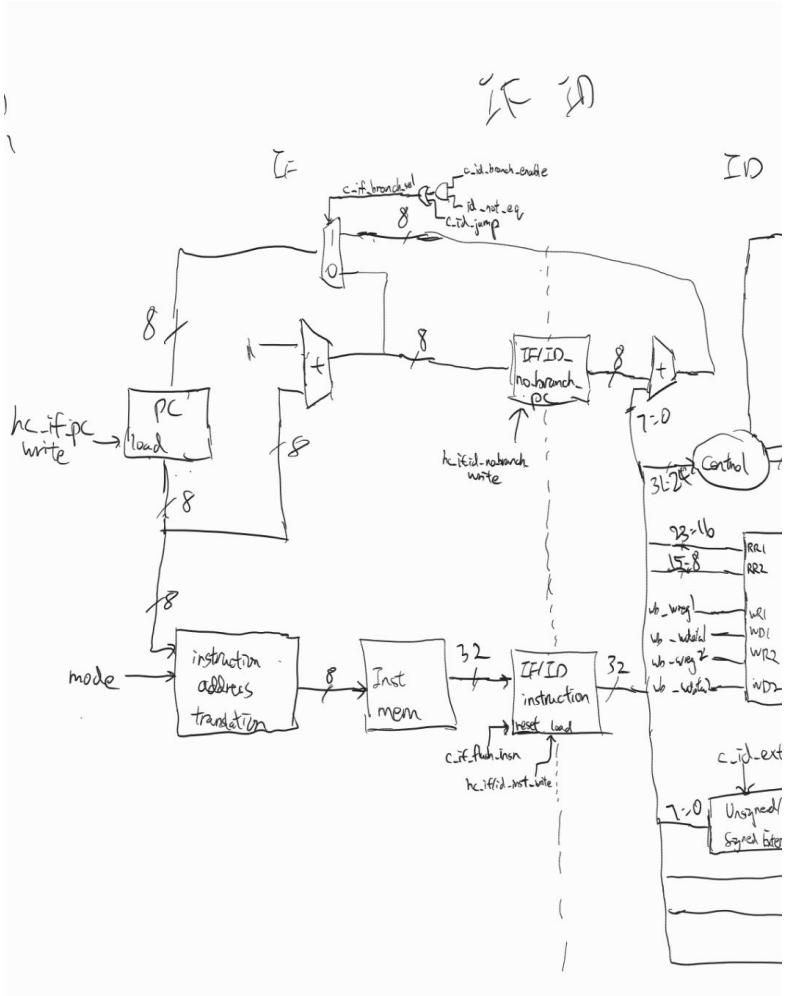
mode selects the set of instruction that perform these functions



Address space for functions



IF stage


IF
ID
c_if_branch_sel
c_id_branch_enable
c_id_jump
id_inst_eq
8
PC
load
hc_if_pc write
IF/ID no branch pc
hc_ifid_nobranch write
Control
mode
instruction address translation
Inst mem
32
IF/ID instruction
reset
load
c_if_flush_insn
hc_ifid_inst_write
c_id_ext
Unsigned/ Signed Extender
RR1
RR2
WR1
WD1
WR2
WD2

MEM stage

STB R1 X X
PNReg <- R1

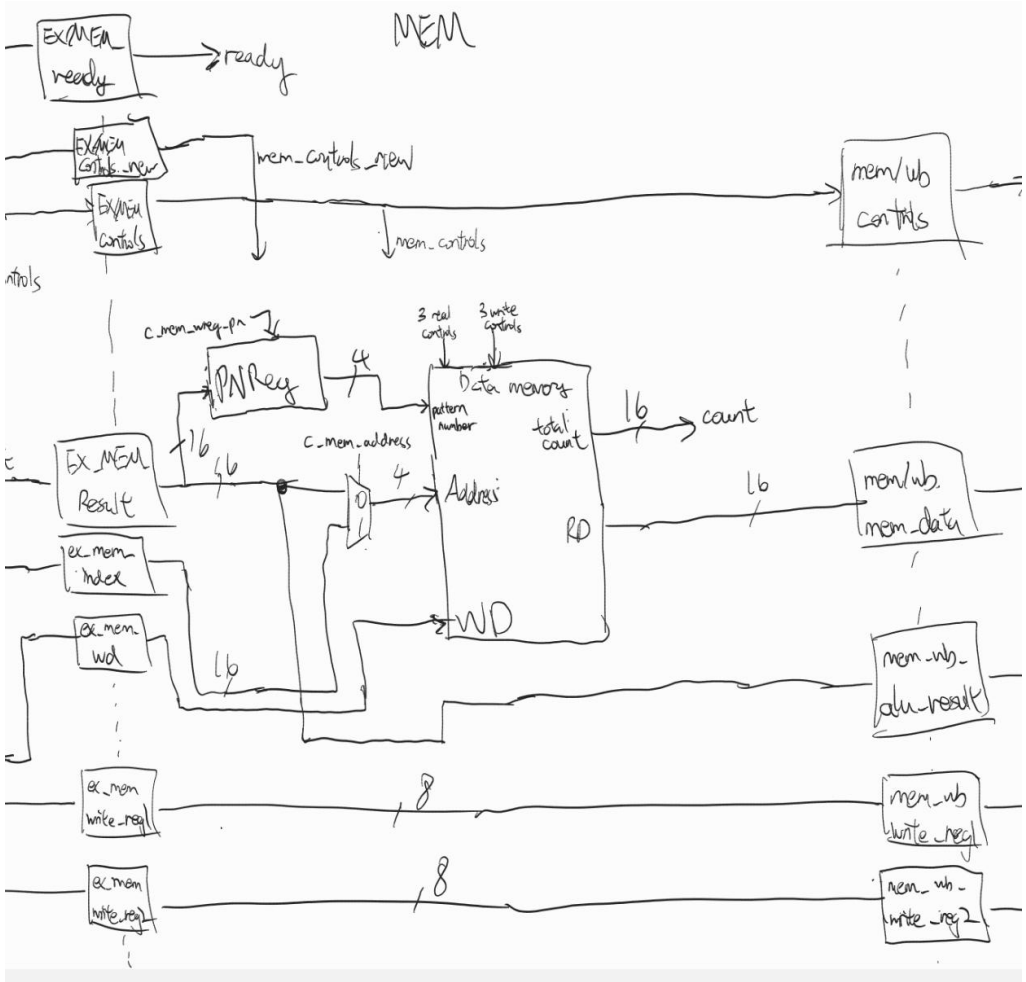
LPA R1 R2 R3

if (R2 < R1)

{R3 <- PA[offset + R2] and R2++}

else

{R3 <- -1}



Data Unit

P0: a
P1: 12
P2 : abc

Example get 2nd char from 3rd pattern:

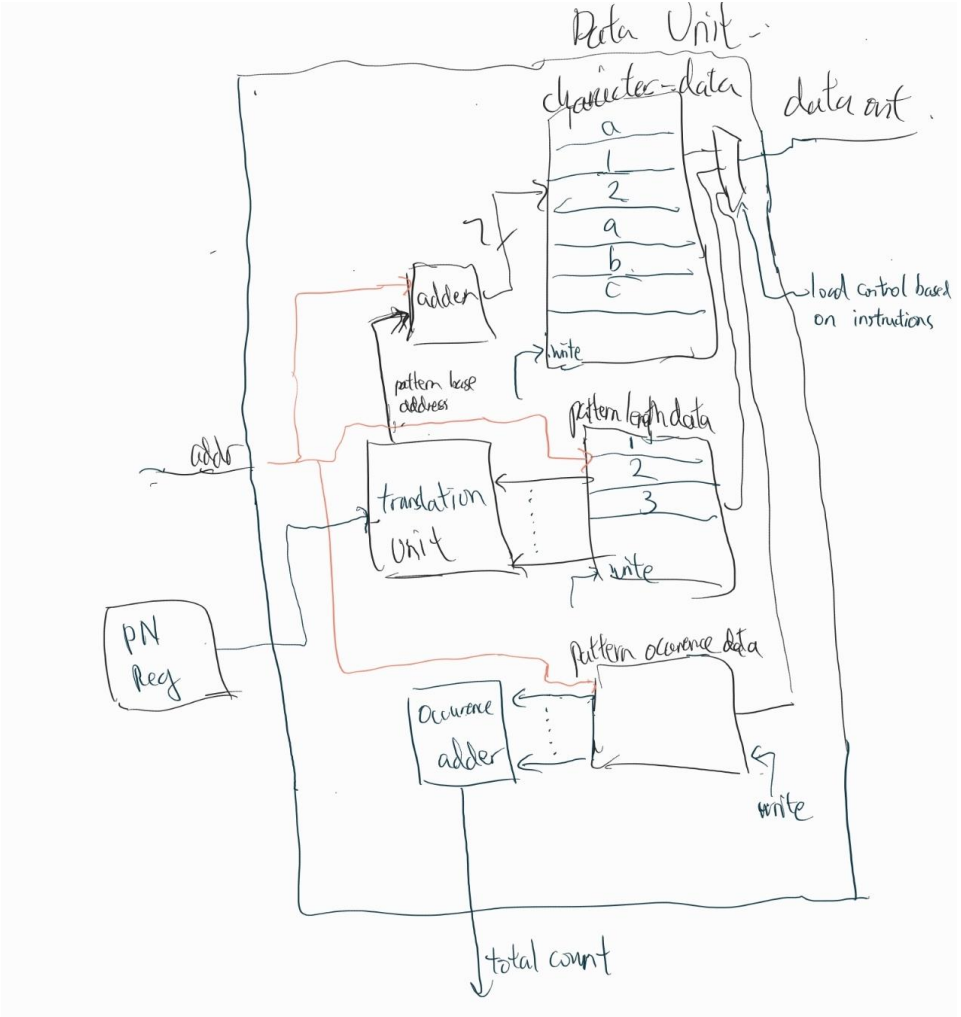
$8 = 2 $11 = b

$9 = 3

$10 = 1

STB $8 X X

LPA $9 $10 $11



Unit Testing: LOA(LOA R1 R2 R3 if R2 < R1, R3 <- OA[R2] and R2++, else R3 <- -1)

LOA Instruction memory:

```
begin
    if (reset = '1') then
        var_insn_mem := (others => X"00000000");
        var_insn_mem(0):= X"090a0b0c";
        var_insn_mem(1):= X"02000000";
```

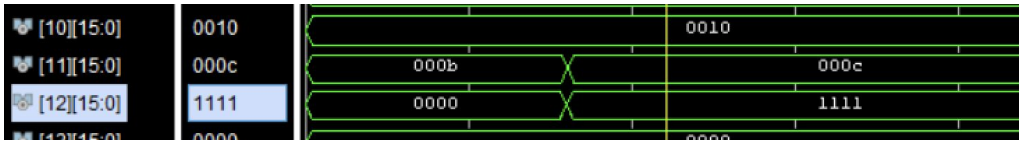
Occurrence memory data:

```
    -- initial values of the data memory : reset to ze
    var_data_mem := (others => X"0000");
    var_data_mem(11) := X"1111";
```

Register file data:

```
-- test
var_regfile(10) := X"0010";
var_regfile(11) := X"000b";
```

Simulation outcome:



Unit Testing: LLA(LLA R1 R2 R3 if R2 < R1, R3 <- LA[R2] and R2++, else R3 <- -1)

LLA Instruction memory:

```
0  if (reset = '1') then
1      var_insn_mem := (others => X"00000000");
2      var_insn_mem(0):= X"070a0b0c";
3      var_insn_mem(1):= X"02000000";
```

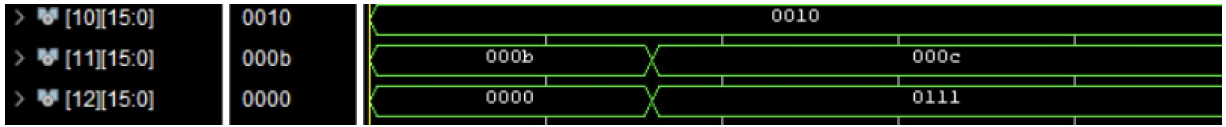
Length memory data:

```
-- test
var_regfile(10) := X"0010";
var_regfile(11) := X"000b";
```

Register file data:

```
84     if (reset = '1') then
85         -- initial values of the data memory : reset to zero
86         var_data_mem := (others => X"0000");
87         var_data_mem(11):= X"0111";
```

Simulation outcome:



Unit Testing: MVI(MVI R1 X R2, R2 <- reg_file[R1])

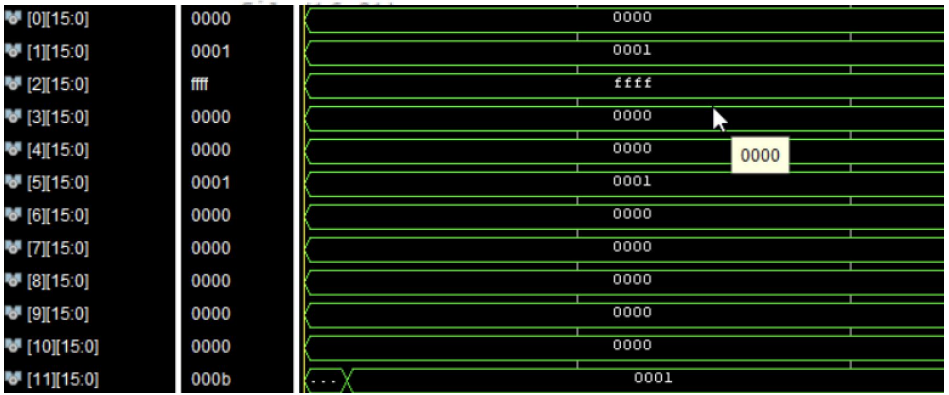
MVI Instruction memory:

```
if (reset = '1') then
    var_insn_mem := (others => X"00000000");
    var_insn_mem(0):= X"0b05000b";
    var_insn_mem(1):= X"0d000000";
```

Register file data:

```
-- test
var_regfile(5) := X"0001";
var_regfile(11) := X"000b";
```

Simualtion outcome:



Unit Testing: SOW (SOW R1 R2 imm, POA[R1+imm] <- R2)

SOW Instruction memory:

```
if (reset = '1') then
    var_insn_mem := (others => X"00000000");
    var_insn_mem(0):= X"020706ff";
    var_insn_mem(1):= X"0d000000";
```

Register file data:

```
    var_regfile(7) := X"0006";
    var_regfile(6) := X"000b";
```

Simulation outcome:

Unit Testing: JP(JP X X imm, jump to PC + 1 + offset)

JP Instruction memory:

```
if (reset = '1') then
    var_insn_mem := (others => X"00000000");
    var_insn_mem(0):= X"0e070002";
    var_insn_mem(1):= X"0d000000";
    var_insn_mem(2):= X"07060708";
    var_insn_mem(3):= X"09060708";
    var_insn_mem(4):= X"0d000000";
```

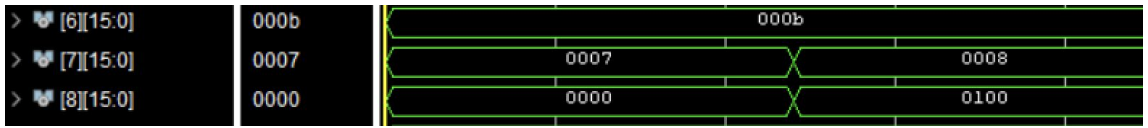
Occurrence memory data:

```
    var_data_mem := (others => X"0000");
    var_data_mem(7) := X"0100";
```

Length memory data:

```
    var_data_mem := (others => X"0000");
    var_data_mem(7):= X"0010";
```

Simulation outcome:



Testing

character array

```
var_data_mem(0) := X"0001";
var_data_mem(1) := X"0001";
var_data_mem(2) := X"0002";
var_data_mem(3) := X"0001";
var_data_mem(4) := X"0002";
var_data_mem(5) := X"0001";
var_data_mem(6) := X"0002";
```

length mem

```
var_data_mem(0):= X"0001";
var_data_mem(1):= X"0003";
var_data_mem(2):= X"0003";
```

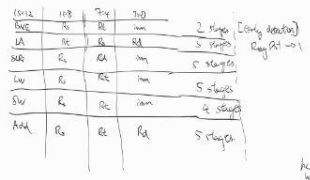
input string

```
var_regfile(16) := X"0001";
var_regfile(17) := X"0002";
var_regfile(18) := X"0001";
var_regfile(19) := X"0002";
var_regfile(20) := X"0001";
```

input string: 1 2 1 2 1
P0: 1
P1: 1 2 1
P2: 2 1 2

3+2+1 = 6

```
        var_insn_mem := (others => X"00000000");
        -- label 1
        var_insn_mem(0):= X"08000006"; -- ADD $0  $0  $6    curr_pattern_num <- 0
        -- label 2
        var_insn_mem(1):= X"04060000"; -- STB $6  0   0     special_p_register <- curr_pattern_num
        var_insn_mem(2):= X"08000007"; -- ADD $0  $0  $7    curr_pattern_offset <- 0
        var_insn_mem(3):= X"080b0308"; -- ADD $11 $3  $8    curr_input_base <- input_register_base + chars_processed
        var_insn_mem(4):= X"07090605"; -- LLA $9  $6  $5    curr_pattern_length <- LA[curr_pattern_num], curr_pattern_num++
        -- label 3
        var_insn_mem(5):= X"0808070c"; -- ADD $8  $7  $12   curr_input_index <- curr_input_base + curr_pattern_offset
        var_insn_mem(6):= X"00000000";
        var_insn_mem(7):= X"00000000";
        var_insn_mem(8):= X"00000000";
        var_insn_mem(9):= X"0b0c000d"; -- MVI $12 0   $13   i_char <- REG[curr_input_index]
        var_insn_mem(10):= X"05050704"; -- LPA $5  $7  $4    p_char <- PA[p_offset+curr_pattern_offset], curr_pattern_offset++
        var_insn_mem(11):= X"060d040a"; -- BNE $13 $4  label 6    if (i_char != p_char) goto label 6
        -- label 4
        var_insn_mem(12)  := X"060705f8";   -- BNE $7  $5  label 3    if (curr_pattern_offset != curr_pattern_length) goto label 3:
        var_insn_mem(13) := X"0f060fff"; -- LOW $6  $15 -1  curr_occurence <- OA[--curr_pattern_num]
        var_insn_mem(14) := X"080f010f"; -- ADD $15 $1  $15     curr_occurence++
        var_insn_mem(15) := X"02060fff"; -- SOW $6  $15 -1  OA[--curr_pattern_num] <- curr_occurence
        -- label 5
        var_insn_mem(16) := X"060609f0"; -- BNE $6  $9  label 2    if (curr_pattern_num != total_number_of_patterns) goto label 2
        var_insn_mem(17) := X"08030103"; -- ADD $3  $1  $3    char_process++
        var_insn_mem(18) := X"06030aed"; -- BNE $3  $10   label 1   if (chars_processed != input_string_length) goto label 1
        var_insn_mem(19) := X"0d000000"; -- END 0   0   0
        var_insn_mem(20) := X"00000000"; -- nop
        var_insn_mem(21) := X"00000000"; -- nop
        -- label 6
        var_insn_mem(22) := X"060e0df9"; -- BNE $14 $13 label 5    if (? != i_char) goto label 5
        var_insn_mem(23) := X"0e0000f4"; -- JP  0     0 laebl 4    jump to label 4
    else
```



15:12	11:8	7:4	3:0	
ONE	Rs	Rt	imm	2 stages [Early detection] Reg Rt -> 1
LA	Rt	Rs	Rd	5 stages
SUB	Rs	Rt	Rd	5 stages
LW	Rs	Rt	imm	5 stages
SW	Rs	Rt	imm	4 stages
ADD	Rs	Rt	Rd	5 stages