# Appendix C – Testing Program

**Listing 1. Program for Branch Prediction Benchmark**

```
1-   add     $t0,$zero,10
2-   loop:
3-   addi    $t1,$t1,1
4-   add     $t2,$zero,$zero
5-   loop2:
6-   addi    $t2,$t2,1
7-   beq     $t0,$t2,exloop2
8-   j       loop2
9-   exloop2:
10-  beq     $t0,$t1,exit
11-  j       loop
12-  exit:
```

**Listing 2. Program for Jump Flush Reduction Benchmark**

```
1-   j    jump1
2-   jump2:
3-   j    jump3
4-   jump1:
5-   j   jump2
6-   jump4:
7-   j    exit
8-   jump3:
9-   j    jump4
10-  exit:
```

**Listing 3. Program for Zero-Stall Benchmark**

```
1-   addi    $t9,$zero,20
2-   loop:
3-   lw      $t1,0($t0)
4-   addi    $t1,$t1,1
5-   addi    $t0,$t0,4
6-   sw      $t1,0($t0)
7-   addi    $t8,$t8,1
8-   beq     $t8,$t9,exit
9-   j       loop
10-  exit:
```

**Listing 4. Program for Bubble Sort Benchmark**

```
1-   NO operation
2-   NO operation
3-   LW  R1, 388(R0)
4-   LW  R2, 384(R0)
5-   LW  R3, 392(R0)
6-   ADD R5,R1,R2
7-   SUB R5,R5,R3
8-   ADD R6,R2,R0
9-   ADD R7,R6,R0
10-  ADD R8,R7,R3
11-  LW  R10, 0(R7)
12-  LW  R11, 0(R8)
13-  SLT R9,R10,R11
14-  BEQ R9,R0, +2
15-  SW  R10, 0(R8)
16-  SW  R11, 0(R7)
17-  ADD R7,R7,R3
18-  BEQ R7,R5, +1
19-  J   9
20-  SUB R5,R5,R3
21-  BEQ R5,R2, +1
22-  J   8
```

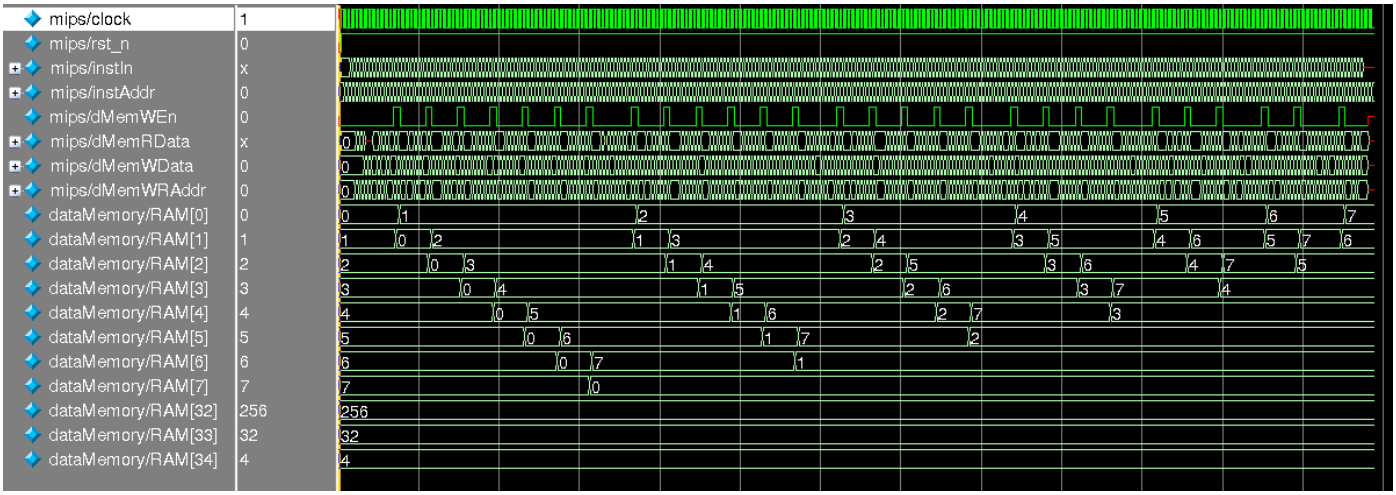



**Figure 1 Bubble Sort Simulation Waveform**

**Listing 5. Program for Sokoban Game**

```
1- .data 0x10010000
2-
3- ##initialize memory with states
4-
5- tile: .word 0x00000004
6-       .word 0x00000004
7-       .word 0x00000004
8-       .word 0x00000004
9-       .word 0x00000004
10-      .word 0x00000004
11-      .word 0x00000004
12-      .word 0x00000004
13-      .word 0x00000004
14-      .word 0x00000004
15-      .word 0x00000000
16-      .word 0x00000000
17-
18-      .word 0x00000004
19-      .word 0x00000000
20-      .word 0x00000000
21-      .word 0x00000000
22-      .word 0x00000000
23-      .word 0x00000000
24-      .word 0x00000000
25-      .word 0x00000000
26-      .word 0x00000000
27-      .word 0x00000004
28-      .word 0x00000000
29-      .word 0x00000000
30-
31-      .word 0x00000004
32-      .word 0x00000000
33-      .word 0x00000000
34-      .word 0x00000002
35-      .word 0x00000006
36-      .word 0x00000002
37-      .word 0x00000000
38-      .word 0x00000002
39-      .word 0x00000000
40-      .word 0x00000004
41-      .word 0x00000000
42-      .word 0x00000000
43-
44-      .word 0x00000004
45-      .word 0x00000004
```

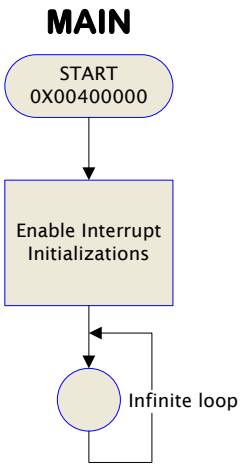
```
46-      .word 0x00000000
47-      .word 0x00000004
48-      .word 0x00000004
49-      .word 0x00000004
50-      .word 0x00000004
51-      .word 0x00000004
52-      .word 0x00000000
53-      .word 0x00000004
54-      .word 0x00000000
55-      .word 0x00000000
56-
57-      .word 0x00000004
58-      .word 0x00000000
59-      .word 0x00000000
60-      .word 0x00000000
61-      .word 0x00000001
62-      .word 0x00000000
63-      .word 0x00000001
64-      .word 0x00000000
65-      .word 0x00000001
66-      .word 0x00000004
67-      .word 0x00000000
68-      .word 0x00000000
69-
70-      .word 0x00000004
71-      .word 0x00000000
72-      .word 0x00000000
73-      .word 0x00000000
74-      .word 0x00000000
75-      .word 0x00000000
76-      .word 0x00000000
77-      .word 0x00000000
78-      .word 0x00000000
79-      .word 0x00000004
80-      .word 0x00000000
81-      .word 0x00000000
82-
83-      .word 0x00000004
84-      .word 0x00000004
85-      .word 0x00000004
86-      .word 0x00000004
87-      .word 0x00000004
88-      .word 0x00000004
89-      .word 0x00000004
90-      .word 0x00000004
91-      .word 0x00000004
92-      .word 0x00000004
93-      .word 0x00000000
94-      .word 0x00000000
95-
96-      .word 0x00000000
97-      .word 0x00000000
98-      .word 0x00000000
99-      .word 0x00000000
100-     .word 0x00000000
101-     .word 0x00000000
102-     .word 0x00000000
103-     .word 0x00000000
104-     .word 0x00000000
105-     .word 0x00000000
106-     .word 0x00000000
107-     .word 0x00000000
108-
109-     .word 0x00000000
110-     .word 0x00000000
111-     .word 0x00000000
112-     .word 0x00000000
```

```
113-     .word 0x00000000
114-     .word 0x00000000
115-     .word 0x00000000
116-     .word 0x00000000
117-     .word 0x00000000
118-     .word 0x00000000
119-     .word 0x00000000
120-     .word 0x00000000
121-
122-     ## START of Text Segment
123-     .text
124-     .globl main
125-
126-     main:
127-                 ## Interrupt Initializations ##
128-                 addi  $t0,$zero,7953          # $t0 = 0x00001f11
129-                 mtc0  $t0,$12                 # Input Status Register
130-
131-                 #put char position offset in t0, depends on level
132-                 addi  $t0,$zero,112
133-                 #put constants in saved registers
134-                 addi  $s1,$zero,1
135-                 addi  $s2,$zero,2
136-                 addi  $s3,$zero,3
137-                 addi  $s4,$zero,4
138-                 addi  $s5,$zero,5
139-                 addi  $s6,$zero,6
140-                 addi  $s0,$zero,11
141-                 ## End Initializations ##
142-
143-     loop: # Infinite loop, wait for interrupt
144-                 j     loop
145-
146-
147-     ## START of Interrupt Segment
148-     .ktext 0x80000180
149-
150-     ## Exception/Interrupt Initializations ##
151-                 add   $k1, $at, $zero         # Save $at register
152-
153-                 mfc0  $k0, $13                # Move Cause into $k0
154-                 srl   $a0, $k0, 2             # Extract ExcCode field
155-                 andi  $a0, $a0, 0xf
156-                 slt   $s7,$s0,$a0
157-                 beq   $s7,$s1,bump
158-     # Branch if ExcCode is Arithmetic OV 12
159-
160-     ## Main Game Logic ##
161-
162-
163-                 add         $t1,$a0,$zero
164-                 # get input from cause reg to t1 reg
165-                 lw          $t9,tile($t0)
166-
167-                 beq         $t1,$s1,up              # check which input
168-                 beq         $t1,$s2,down
169-                 beq         $t1,$s3,left
170-                 beq         $t1,$s4,right           # t1=state of input
171-
172-
173-     up:
174-                 addi  $t2,$t0,-48             # t2=offset of next tile
175-                 lw          $t3,tile($t2)     # t3=state of next tile
176-                 beq         $t3,$s2,upnext    # check if next to box
177-                 beq         $t3,$s3,upnext    # check if next to box on target
178-                 beq         $zero,$zero,check
179-           upnext:
```

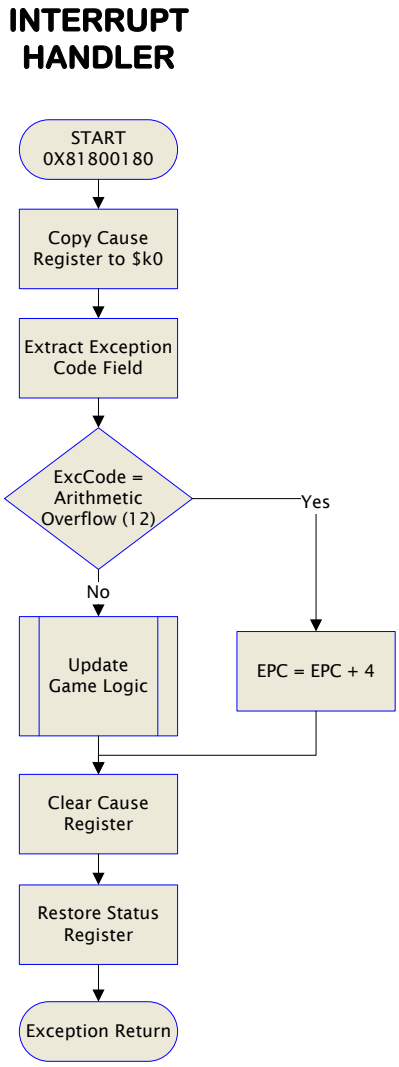
```
180-                addi  $t4,$t0,-96             # t4=offset of next to next tile
181-                lw          $t5,tile($t4)     # t5=state of next to next tile
182-                beq         $zero,$zero,check
183-
184-    down:
185-                addi        $t2,$t0,48
186-                lw          $t3,tile($t2)
187-                beq         $t3,$s2,downnext
188-                beq         $t3,$s3,downnext
189-                beq         $zero,$zero,check
190-          downnext:
191-                addi        $t4,$t0,96
192-                lw          $t5,tile($t4)
193-                beq         $zero,$zero,check
194-
195-    left:
196-                addi        $t2,$t0,-4
197-                lw          $t3,tile($t2)
198-                beq         $t3,$s2,leftnext
199-                beq         $t3,$s3,leftnext
200-                beq         $zero,$zero,check
201-          leftnext:
202-                addi        $t4,$t0,-8
203-                lw          $t5,tile($t4)
204-                beq         $zero,$zero,check
205-
206-    right:
207-                addi        $t2,$t0,4
208-                lw          $t3,tile($t2)
209-                beq         $t3,$s2,rightnext
210-                beq         $t3,$s3,rightnext
211-                beq         $zero,$zero,check
212-          rightnext:
213-                addi        $t4,$t0,8
214-                lw          $t5,tile($t4)
215-                beq         $zero,$zero,check
216-
217-    check:
218-                beq         $t3,$zero,shift
219-    # if next to char is empty space, goto shift
220-                beq         $t3,$s1,shift2
221-    # if next to char is target, goto shift2
222-                beq         $t3,$s2,checknext
223-    # if next to char is box, goto check next tile
224-                beq         $t3,$s3,checknext
225-    # if next to char is box on target, check next tile
226-
227-                beq         $zero,$zero,no_move
228-    shift:
229-                sw          $s6,tile($t2)           # put char in target tile
230-                beq         $t9,$s5,leave_target
231-    # if char on top of target, goto leave_target
232-
233-                sw          $zero,tile($t0)
234-                beq         $zero,$zero,exit_input
235-
236-    shift2:
237-                sw          $s5,tile($t2)
238-                beq         $t9,$s5,leave_target
239-                sw          $zero,tile($t0)
240-                beq         $zero,$zero,exit_input
241-
242-
243-    checknext:
244-                beq         $t5,$zero,shiftbox
245-                beq         $t5,$s1,shifttgt
246-                beq         $zero,$zero,no_move
```

```
247-
248-          shiftbox:
249-
250-                sw          $s2,tile($t4)
251-     # store box in next-next tile
252-                beq         $t3,$s3,leaveboxtgt_1
253-     # if next to char is box on target, goto leaveboxtgt_1
254-                sw          $s6,tile($t2)
255-     # store char in next tile
256-                beq         $zero,$zero,shiftbox_char
257-                leaveboxtgt_1:
258-                sw          $s5,tile($t2)
259-
260-          shiftbox_char:
261-                beq         $t9,$s5,leave_target
262-                sw          $zero,tile($t0)
263-                beq         $zero,$zero,exit_input
264-
265-          shifttgt:
266-                sw          $s3,tile($t4)
267-                beq         $t3,$s3,leaveboxtgt_2
268-                sw          $s6,tile($t2)
269-                beq         $zero,$zero,shiftbox_char2
270-                leaveboxtgt_2:
271-                sw          $s5,tile($t2)
272-
273-          shiftbox_char2:
274-                beq         $t9,$s5,leave_target
275-                sw          $zero,tile($t0)
276-                beq         $zero,$zero,exit_input
277-
278-
279-     leave_target:
280-                sw          $s1,tile($t0)
281-
282-     exit_input:
283-                move  $t0,$t2                 # set new offset for character
284-     no_move:
285-
286-                mfc0  $k0, $14                # Load EPC to $k0
287-                beq         $zero,$zero,done
288-     bump:
289-
290-     ## Exception/Interrupt Exit Routine ##
291-
292-                mfc0  $k0, $14                # Load EPC to $k0
293-                addiu $k0, $k0, 4
294-     # Do not reexecute faulting instruction
           # EPC = EPC + 4
295-                mtc0  $k0, $14                # Move New EPC to CoProc
296-
297-     done:
298-                mtc0  $0, $13                 # Clear Cause register
299-
300-                mfc0  $k0, $12                # Restore Status register
301-                addi  $k0,$zero,7953          # $t0 = 0x00001f11
302-                mtc0  $k0, $12                # Enable interrupts again
303-
304-                add         $at, $k1,$zero
305-                eret                          # Return to address in EPC
```

### A. Sokoban Implementation

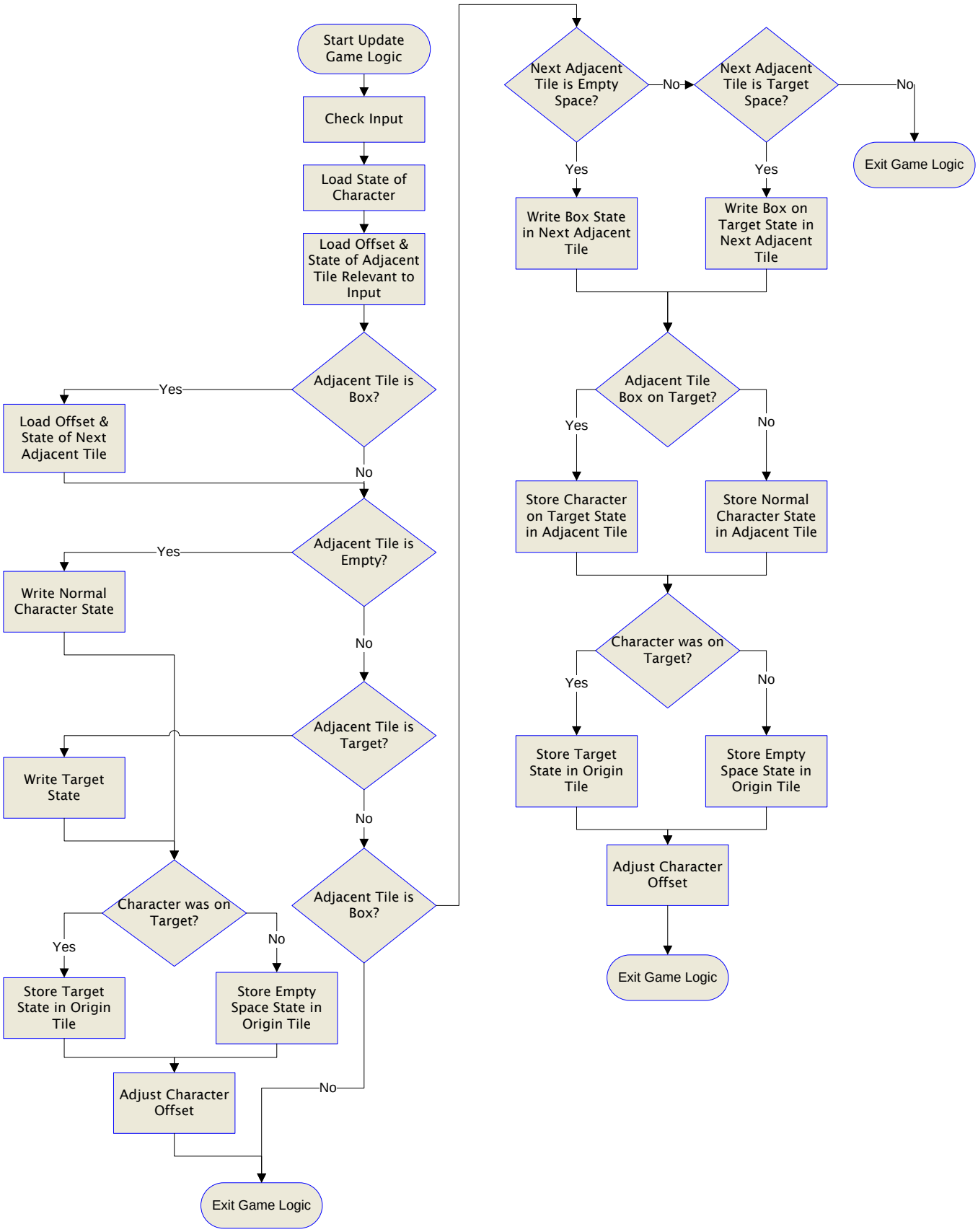



The main function initializes registers and enables interrupt. It then infinitely loops waiting for external interrupt. It may also check for the winning condition.




The Interrupt Handler contains an arithmatic overflow exception handler as it is a basic feature of a handler routine. The handler increments the EPC to make the processor skip the fault instruction. If the interrupt originates from the input, the processor will execute game logic subroutine.

# GAME LOGIC




The game logic subroutine main function is to update the data memory according to the input. The game logic specifies the rules of the game. It always loads the current state of the character, the location of the character, the location (in memory) and state (value) of adjacent tile relevant to input, and if needed, the next adjacent tile to registers.