Programming Project Checkpoint 5

[112006234] [Justin Thiadi 程煒財]

Compilation of Code:

```
User@LAPTOP-U8BRQGS7 /cygdrive/C/Users/User/Documents/OS/112006234_ppc5
$ make clean
rm *.hex *.ihx *.lnk *.lst *.map *.mem *.rel *.rst *.sym
rm: cannot remove '*.ihx': No such file or directory
rm: cannot remove '*.lnk': No such file or directory
make: *** [Makefile:18: clean] Error 1

User@LAPTOP-U8BRQGS7 /cygdrive/C/Users/User/Documents/OS/112006234_ppc5
$ make
sdcc -c  --model-small testlcd.c
sdcc -c  --model-small preemptive.c
preemptive.c:86: warning 85: in function ThreadCreate unreferenced function argu
ment : 'fp'
sdcc -c  --model-small lcdlib.c
lcdlib.c:75: warning 85: in function delay unreferenced function argument : 'n'
sdcc -c  --model-small buttonlib.c
sdcc -c  --model-small keylib.c
sdcc  -o testlcd.hex testlcd.rel preemptive.rel lcdlib.rel buttonlib.rel keylib.
rel
```

After testing and verifying that LCD works correctly, I commented out the *testlcd* part in my *makefile* (including *buttonlib* since we won't be using in part 2)

```
User@LAPTOP-U8BRQGS7 /cygdrive/C/Users/user/Documents/OS/112006234_ppc5
$ make clean
rm -f *.hex *.ihx *.lnk *.lst *.map *.mem *.rel *.rst *.sym

User@LAPTOP-U8BRQGS7 /cygdrive/C/Users/user/Documents/OS/112006234_ppc5
$ make
sdcc -c --model-small dino.c
dino.c:198: warning 283: function declarator with no prototype
sdcc -c --model-small preemptive.c
preemptive.c:88: warning 85: in function ThreadCreate unreferenced function argu
ment : 'fp'
sdcc -c --model-small lcdlib.c
lcdlib.c:75: warning 85: in function delay unreferenced function argument : 'n'
sdcc -c --model-small buttonlib.c
sdcc -c --model-small keylib.c
sdcc  -o dino.hex dino.rel preemptive.rel lcdlib.rel buttonlib.rel keylib.rel
```

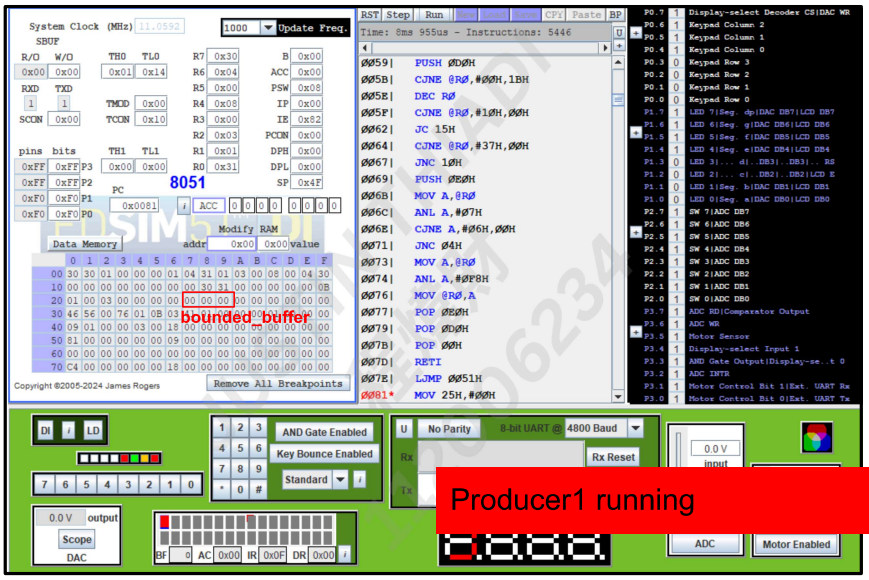
1.1.1. Peripheral Devices:

Value	Global	Global Defined In Module
00000000	.__.ABS.	_gptrget
00000020	_mutex	testlcd
00000021	_full	testlcd
00000022	_empty	testlcd
00000023	_read	testlcd
00000024	_write	testlcd
00000025	_button_in	testlcd
00000026	_key_in	testlcd
00000027	_bounded_buffer	testlcd
0000002A	_output_char	testlcd
00000030	_saved_SP	preemptive
00000034	_current_thread	preemptive
00000035	_valid_thread	preemptive
00000036	_new_thread	preemptive
00000037	_old_SP	preemptive
00000038	_dir	preemptive
0000003B	_button_state	buttonlib
0000003C	_lcd_ready	lcdlib

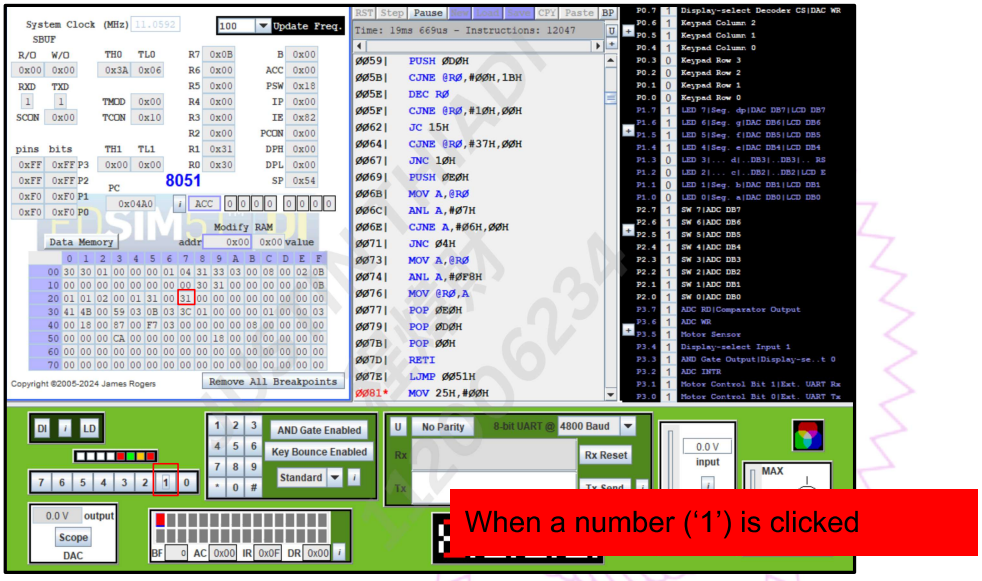
Value	Global	Global Defined In Module
00000081	_Producer1	testlcd
000000C4	_Producer2	testlcd
00000107	_Consumer	testlcd
0000013C	_main	testlcd
00000163	__sdcc_gsinit_startup	testlcd
00000167	__mcs51_genRAMCLEAR	testlcd
00000168	__mcs51_genXINIT	testlcd
00000169	__mcs51_genXRAMCLEAR	testlcd
0000016A	_timer0_ISR	testlcd
00000170	_Bootstrap	preemptive
00000196	_ThreadCreate	preemptive
0000021B	_ThreadYield	preemptive
0000026D	_ThreadExit	preemptive
000002CC	_myTimer0Handler	preemptive
00000330	_LCD_ready	lcdlib
00000334	_LCD_Init	lcdlib
00000347	_LCD_IRWrite	lcdlib
00000369	_LCD_functionSet	lcdlib
00000393	_LCD_write_char	lcdlib
000003BD	_LCD_write_string	lcdlib
000003F2	_delay	lcdlib
000003F6	_AnyButtonPressed	buttonlib
00000407	_ButtonToChar	buttonlib
00000493	_Init_Keypad	keylib
00000499	_AnyKeyPressed	keylib
000004A6	_KeyToChar	keylib
00000522	__gptrget	_gptrget

1.1.2. Button Bank:

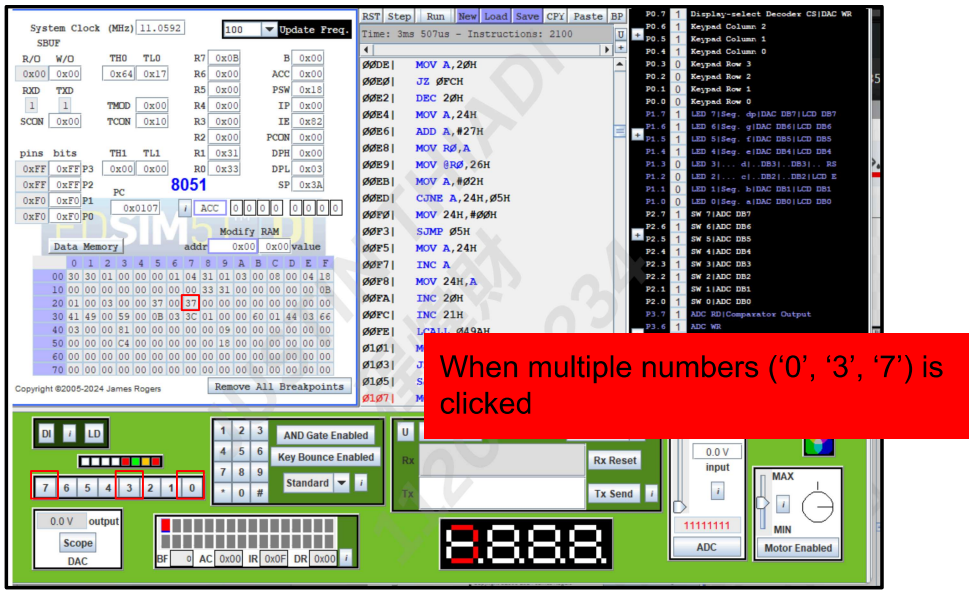
This function is used to poll whether any button is currently being pressed. It returns non-zero if at least one button is pressed.



No buttons pressed (*bounded_buffer* initialized; is still empty)



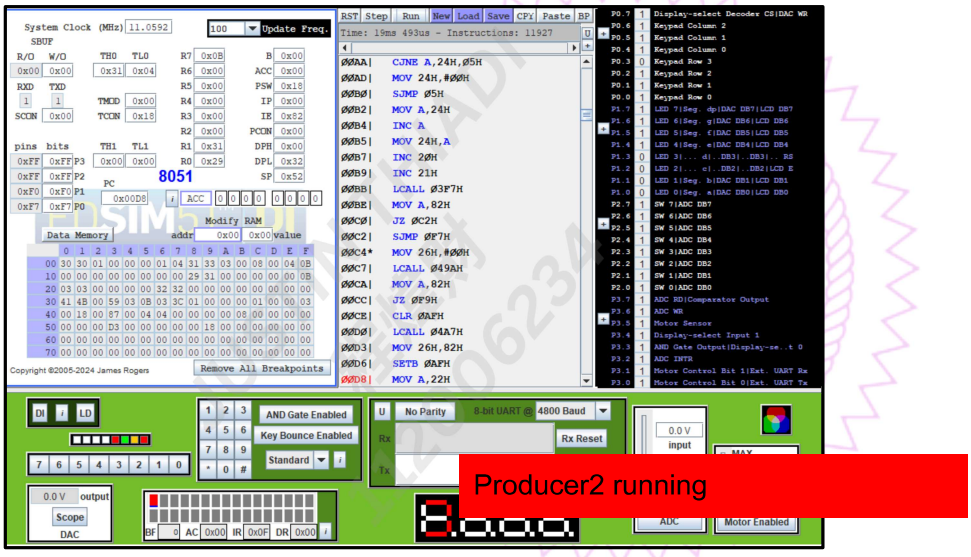
in case multiple buttons are pressed, return the highest numbered button (refer to *buttonlib.c*). In our case, it is ***0x37 (‘7’ ASCII)*** as seen in the screenshot below:

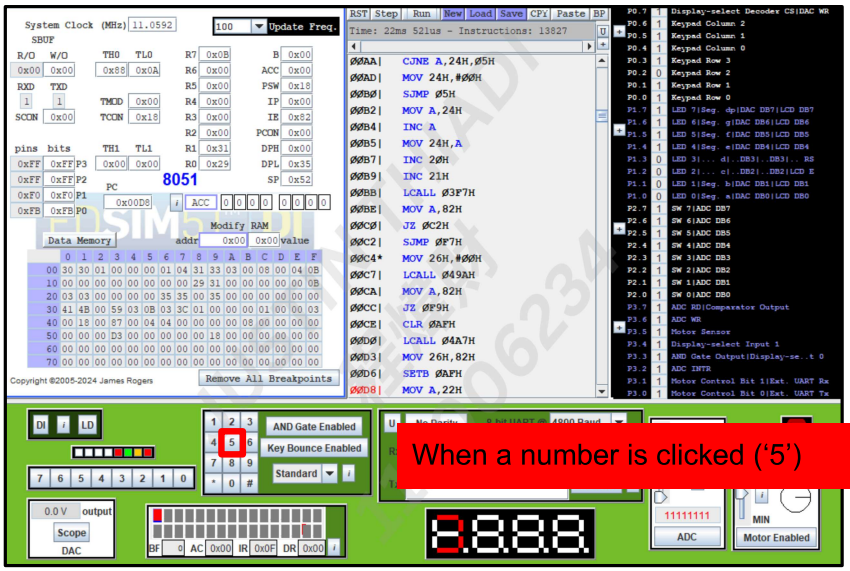


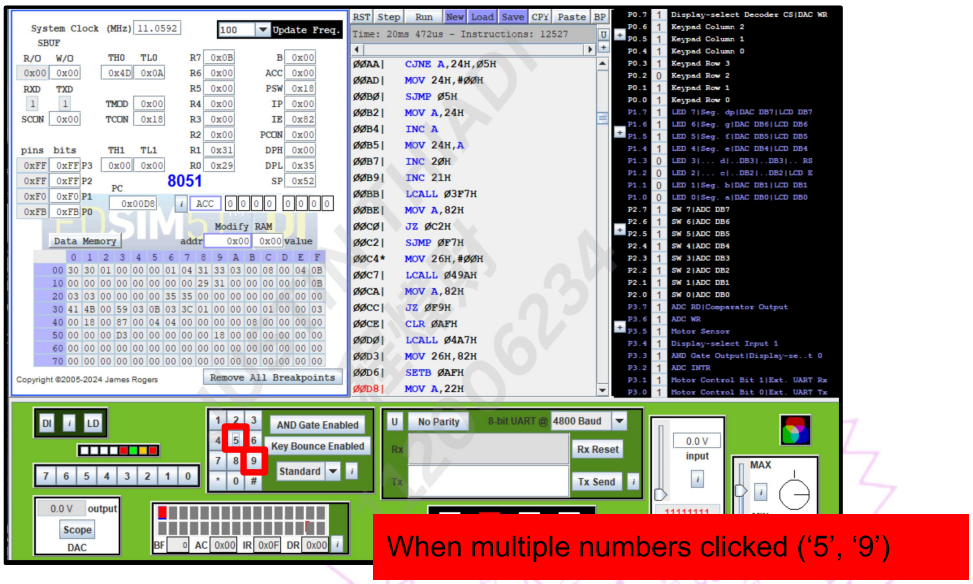
1.1.3. Keypad:

The *AnyKeyPressed()* function correctly detects if any key is pressed.

The *KeyToChar()* function correctly returns the ASCII character for the pressed key, including edge cases like multiple keys being pressed. In other words, it maps row/col to ASCII ('0'–'9', '*', '#').

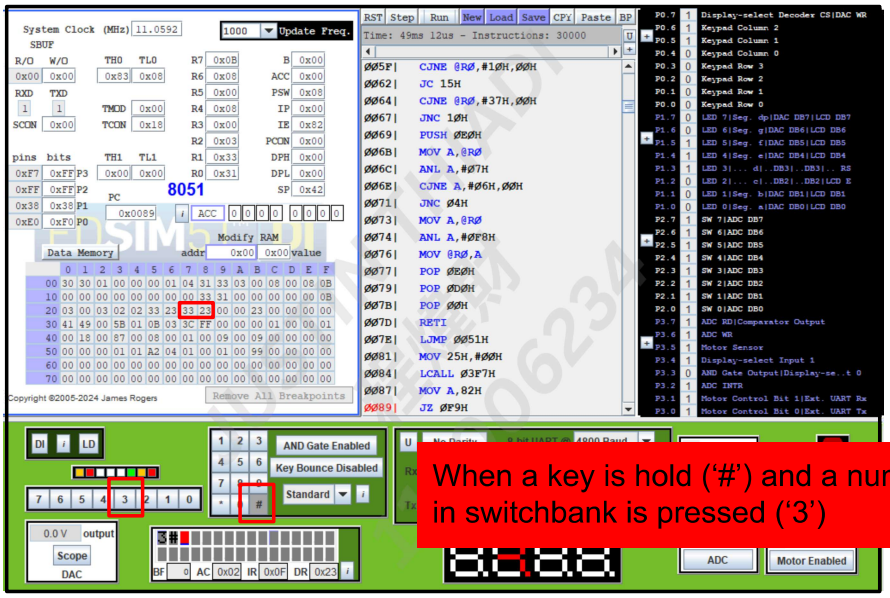






- Hold down two keys ('5' and '9'):
- Only the one with higher priority (as per scan order) should be returned.
- The scan order is: top row → next row → third row → last row.

1.1.4. LCD:



- Initialized by *LCD_Init()*.
- Custom symbols loaded via *LCD_set_symbol(code, bitmap[])*.
- Data writes via *LCD_cursorGoTo(row, col) + LCD_write_char()*.

Press a key on the keypad:

- enqueue one character
- Display on LCD
- No repeat while holding

Press a button from the button bank:

- enqueue and display a character

Press keypad and button simultaneously:

- Both enqueue their own characters
- Order vary depending on thread scheduling

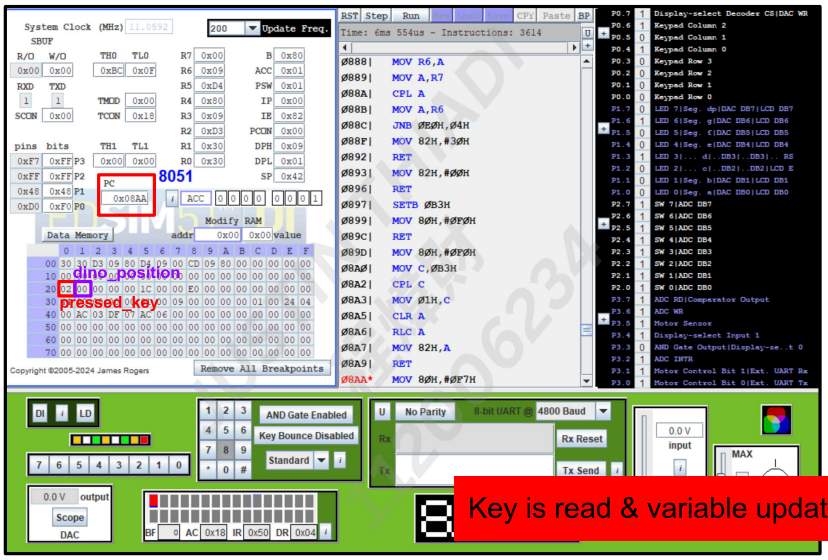
After making sure that the buttons, keypad, and LCD is working well, we can move on to part 2.

1.2.1 Dinosaur Game

```
      Value  Global                           Global Defined In Module
      -----  --------------------------------  ------------------------
C:    00000081  _keypad_ctrl                   dino
C:    000000FB  _render_task                   dino
C:    0000022E  _game_ctrl                     dino
C:    0000039A  _difficulty_input              dino
C:    000003DF  _main                          dino
C:    0000044B  __sdcc_gsinit_startup          dino
C:    0000044F  __mcs51_genRAMCLEAR            dino
C:    00000450  __mcs51_genXINIT               dino
C:    00000451  __mcs51_genXRAMCLEAR           dino
C:    00000452  _timer0_ISR                    dino
C:    00000458  _Bootstrap                     preemptive
C:    0000047E  _ThreadCreate                  preemptive
C:    00000504  _ThreadYield                   preemptive
C:    0000055E  _ThreadExit                    preemptive
C:    000005BD  _myTimer0Handler               preemptive
C:    0000061F  _ThreadReset                   preemptive
C:    00000623  _LCD_ready                     lcdlib
C:    00000627  _LCD_Init                      lcdlib
C:    0000063A  _LCD_IRWrite                   lcdlib
C:    0000065C  _LCD_functionSet               lcdlib
C:    00000686  _LCD_write_char                lcdlib
C:    000006B0  _LCD_write_string              lcdlib
C:    000006E5  _delay                         lcdlib
C:    000006E9  _LCD_set_symbol                lcdlib
C:    000007FA  _AnyButtonPressed              buttonlib
C:    0000080B  _ButtonToChar                  buttonlib
C:    00000897  _Init_Keypad                   keylib
C:    0000089D  _AnyKeyPressed                 keylib
C:    000008AA  _KeyToChar                     keylib
C:    00000926  __moduint                      _moduint
C:    00000973  __gptrget                      _gptrget
C:    0000098F  __modsint                      _modsint

ASxxxx Linker V03.00/V05.40 + sdld,  page 14.
Hexadecimal  [32-Bits]

Area                                Addr       Size        Decimal Bytes (Attributes)
--------------------------------    ----       ----        ------- ----- ------------
CONST                               000009C5   00000016 =          22. bytes (REL,CON,CODE)

      Value  Global                            Global Defined In Module
      -----  --------------------------------  ------------------------
C:    000009C5  _dinosaur                      dino
C:    000009CD  _cactus                        dino
C:    000009D5  _cactus_patterns               dino
```

```
  Value  Global                            Global Defined In Module
  -----  --------------------------------  ------------------------
00000000  .__.ABS.                         _modsint
00000020  _pressed_key                     dino
00000021  _dino_position                   dino
00000022  _game_start                      dino
00000023  _game_over                       dino
00000024  _score                           dino
00000025  _map_top                         dino
00000027  _map_btm                         dino
00000029  _time                            dino
0000002A  _carry_bit                       dino
0000002B  _mask                            dino
0000002C  _game_speed                      dino
0000002D  _cactus_timer                    dino
0000002E  _last_cactus_row                 dino
00000030  _saved_SP                        preemptive
00000034  _current_thread                  preemptive
00000035  _valid_thread                    preemptive
00000036  _new_thread                      preemptive
00000037  _old_SP                          preemptive
00000038  _reload_time                     preemptive
0000003B  _button_state                    buttonlib
0000003C  _lcd_ready                       lcdlib
```

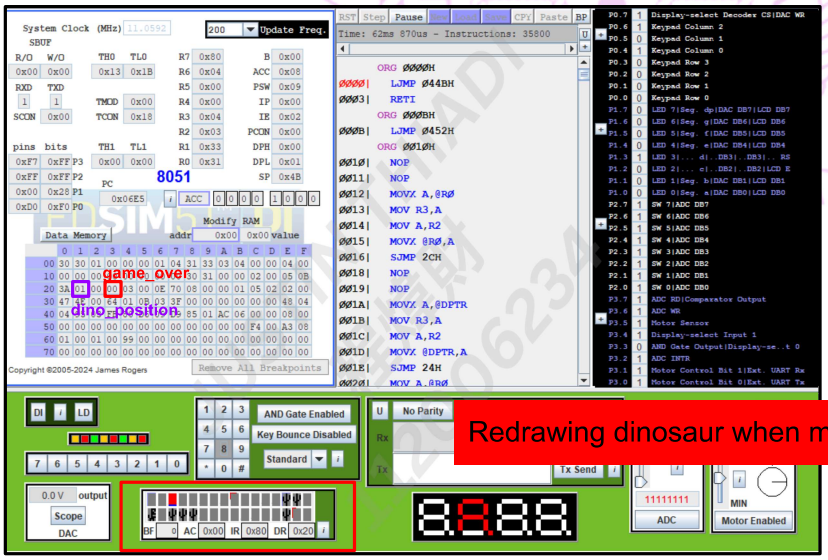
1.2.2 keypad_ctrl



- Waits for *AnyKeyPressed()*, then disables interrupts (*EA=0*).
- Reads *pressed_key = KeyToChar(),* re-enables interrupts (*EA=1*).
- On '2'/'8', in a protected block updates dino_position, checks collision *(map_top[0]/map_btm[0])*, sets *game_over*, and redraws the dinosaur after the difficulty is set by pressing the keypad (number + #)

Observations:

- *pressed_key* (showing '2')
- *dino_position* (showing 0)
- *PC* pointing at *0x8AA* (which is the memory address for *pressed_key*).
- *pressed_key* loaded the correct ASCII code
- The old vs. new *dino_position* value is still unchanged here

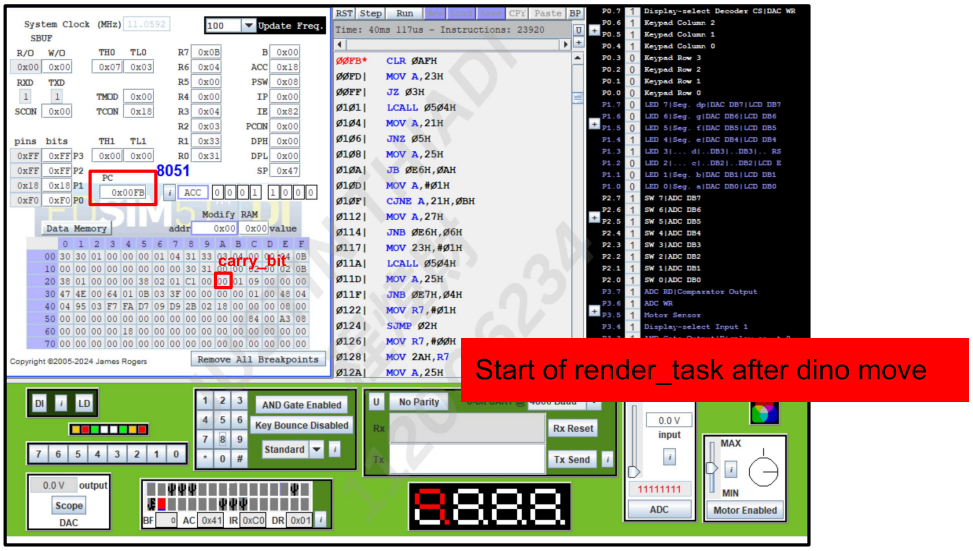


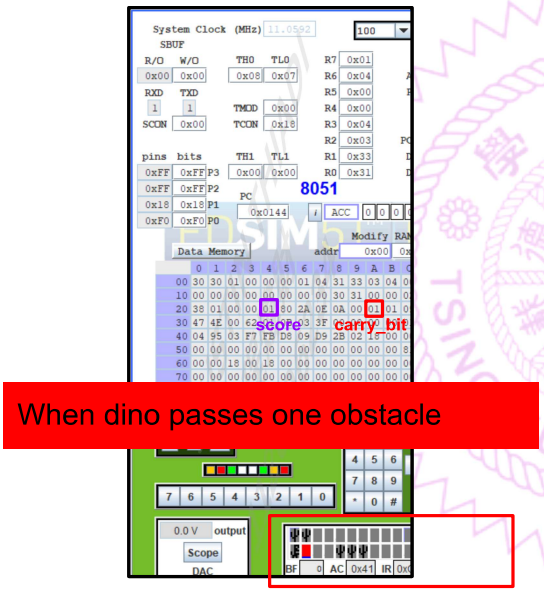
LCD window showing:

- The old dinosaur removed from the previous row
- The new dinosaur in the opposite row, column 0

no cactus collided during this time (*game_over* is still 0)

1.2.3 render_task





Disables interrupts, **checks** *game_over*.

Shifts *map_top[]* and *map_btm[]* left by one bit, tracks carry to increment score and wrap bits between bytes.

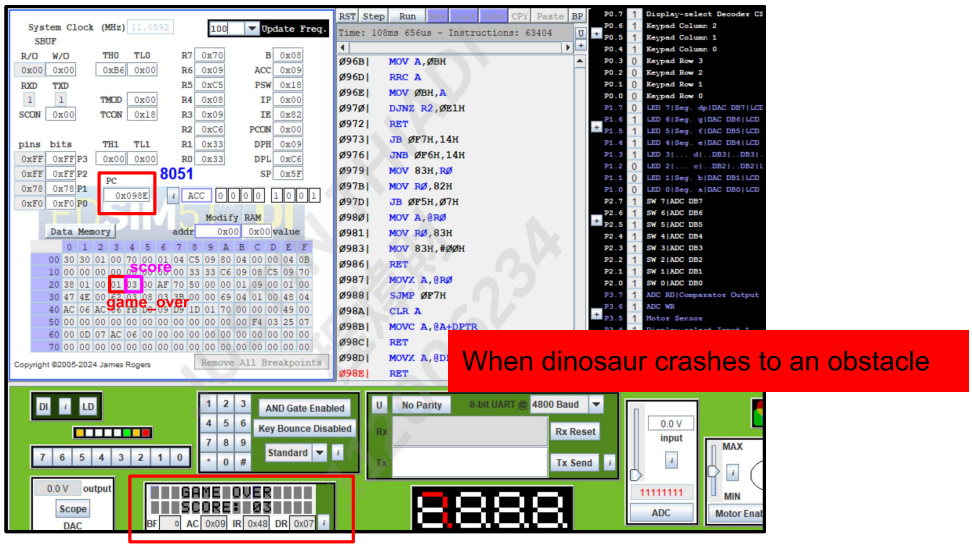
- carry_bit shifted out and the score++ (from 0 to 1).

Re-enables interrupts, then redraws the entire 2×16 map on the LCD by looping on each bit and writing the cactus (\2).

Then draws the dinosaur at dino_position.

Note that map_top and map_bottom values are also modified accordingly with the shifting of the cacti based on difficulty, and movement of the dinosaur (for simplicity, it is not shown in the screenshot but increases between the values can be seen by comparing from the previous screenshots)

1.2.4 game_ctrl



- Runs in *main* after thread creation.
- Increments *cactus_timer* each cycle
  - when it reaches *(10 - game_speed)*:
    - resets timer
    - selects a pattern from *cactus_patterns[],* creates a new cactus by OR-ing the low bit of *map_top[1]* or *map_btm[1]*
    - updates *last_cactus_row*.
- Loops *ThreadYield()* game_speed times to control frame rate.
- LCD window showing the full “GAME OVER” text on row 0 and the two-digit score on row 1 (with correct spaces as in the code).
  - *game_over* == 1
  - *score* still holding final value (3 in this case)
- Then exit *game_ctrl()* - indicated by being in the memory address of *RET* (*0x98E*) which will pop the return address off the stack and jump back into *main()* after the *game_ctrl();* call, which then executes *ThreadReset()*.

1.3 Questions

- **What variables may have race conditions and why?**
  - ***map_top[] & map_btm[]***
    - Both *render_task()* (shifts + score bump) and *game_ctrl()* (bit-OR injection) read/write these two-byte arrays in multiple instructions. Without locking, one thread could preempt in the middle of *SHIFT* or *OR*, leaving the high and low bytes out of sync.
      - How I fixed it: wrapped both the *shift/score++* block in *render_task()* and the *if (cactus_timer ≥ …) { … map_*[1] |= 0x01; last_cactus_row = … } block in game_ctrl()* with

- *EA = 0;*
- *EA = 1;*

ensuring no context switch can occur mid-update.

- ***score***
  - *render_task()* increments score when a cactus shifts off the left edge, while *game_ctrl()* reads score to pick the next injection pattern. An interrupt during *score++* could lose an increment.
  - How I fixed it: The *EA=0/EA=1* around the entire *shift* and *score++* sequence in render_task() also protects score.

- ***game_over***
  - All three threads set or test *game_over* when detecting collisions or shutting down. Without exclusion, two threads could simultaneously write it or one could test it based on old memory.
    - How I fixed it: where there is collision, it now runs with interrupts disabled, so the flag write and the following yield cannot be interrupted.

- **pressed_key & dino_position**
  - *keypad_ctrl()* writes *pressed_key* and updates *dino_position* on each arrow press, while *render_task()* reads both to check for collisions and redraw.
  - How I fixed it: *pressed_key = KeyToChar()* inside *EA=0/EA=1*, and added *EA=0/EA=1* around the entire *if (pressed_key == '2' || '8') { … dino_position = … }* block.

- **What data type do you use for the map?**
  - Use 2 bytes per row (unsigned char map_top[2] and unsigned char map_btm[2]) to represent 16 columns as 16 bits.
    - Each bit = “cactus present” or “empty.”
    - Total memory = 4 bytes for both rows instead of 32 bytes.

- **How do you generate a new cactus?**
  - In *game_ctrl*, I tick a *cactus_timer*. Once it reaches *(10 − game_speed)*:
    - Reset *cactus_timer = 0.*
    - Choose a pattern like
      - *unsigned char pattern = cactus_patterns[(score + last_cactus_row) % sizeof(cactus_patterns)];*
    - *If pattern == 1, do map_top[1] |= 0x01; last_cactus_row = 1;*
    - *else if pattern == 2, do map_btm[1] |= 0x01; last_cactus_row = 2;*
    - *else leave both rows.*

    This approach creates a top/bottom cactus and avoids cacti only one column apart from each or full walls.