<N26112437>_<劉兆軒> AIAS 20223 Lab 2 HW Submission

1. 請不要用這份template 交作業, 建立一個新的codimd 檔案, 然後copy & paste 這個 template 到你創建的檔案做修改。
2. 請修改你的學號與姓名在上面的title 跟這裡, 以避免TA 修改作業時把檔案跟人弄錯了
3. 在Playlab 作業中心繳交作業時, 請用你創建的檔案鏈結繳交, 其他相關的資料與鏈結請依照Template 規定的格式記載於codimd 上。

記得在文件標題上修改你的 <N26112437> <劉兆軒>

Gitlab code link

- https://playlab.computing.ncku.edu.tw:4001/kevin1217/lab02
  (https://playlab.computing.ncku.edu.tw:4001/kevin1217/lab02) -

HW 2-1 Bit Operations

Code

Please copy and paste your code for the following 4 functions in this section.

- get_bit()
- set_bit()
- flip_bit()
- decode_riscv_inst()

Remember to write clear comment in your code to explain your implementation in more details.

get_bit()

假設 `x=0011` ， `n=1` ，則 `mask` =0010，做 `&` 後 `bit` 為0010，可找出該位置的 `bit` 為何，最後 `bit` 歸位get值。

```
// Return the nth bit of x.
// Assume 0 <= n <= 31
unsigned get_bit(unsigned x,
    unsigned n) {

    int mask = 1 << n;
    unsigned bit = x & mask;
    bit = bit >> n;
    return bit;
}
```

set_bit()

考慮兩種情況，當 `v=0` 時， `mask` 用 `&` 覆蓋所需set位置的值，其中 `mask` 需先反轉(如0010->1101)；而當 `v=1` 時， `mask` 用 `|` 覆蓋原有值。

```
// Set the nth bit of the value of x to v.
// Assume 0 <= n <= 31, and v is 0 or 1
void set_bit(unsigned* x,
    unsigned n,
    unsigned v) {

    int mask = 1 << n;
    unsigned bit = *x & mask;
    bit = bit >> n;

    if (v == 0)
    {
        int mask = ~(1 << n);
        *x &= mask;
    }
    else
    {
        int mask = v << n;
        *x |= mask;
    }

}
```

flip_bit()

這個function主要是將指定位置的值去做改變，0變1，1變0，此時同樣用 `mask` 對應指定位置，做 ^ 可以使對應位改變（即該位置若是1，則對到 `mask` 的1變0，若是0，對到 `mask` 的1變1）。

```
// Flip the nth bit of the value of x.
// Assume 0 <= n <= 31
void flip_bit(unsigned* x,
    unsigned n) {
    // YOUR CODE HERE
    int mask = 1 << n;
    *x = *x ^ mask;

}
```

decode_riscv_inst()

假設start為6，end為3，addr為0101 0101，希望得到該addr由右邊開始的第3個數字到第5個數字。

`#define BIT_MASK(n) ((((uint32_t)1)<< ((n) < 32 ? (n) : 31)) -1)` 先透過這段程式得到0011 1111和0000 0111，即n為多少則產生多少1。

`#define BIT_MASK2(start, end) (BIT_MASK(start) & ~BIT_MASK(end))` 再透過這段程式計算0011 1111&1111 1000得到0011 1000，即產生特定位置的mask。

`#define GET_ADDR_BITS(addr, start, end) (((addr) & BIT_MASK2(start, end)) >> (end))` 最

後將0101 0101&0011 1000得到0001 0100，位移end位對齊最低位，即找出該ddr指定位置的bit為何。

有了上述巨集，即可參考每個指令bit的差異，去做篩選找出不同OPCODE對應的指令為何。

```
//Encoding the instruction
OPCODE decode_riscv_inst(uint32_t inst) {

#define BIT_MASK(n) ((((uint32_t)1)<< ((n) < 32 ? (n) : 31)) -1)
#define BIT_MASK2(start, end) (BIT_MASK(start) & ~BIT_MASK(end))
#define GET_ADDR_BITS(addr, start, end)  (((addr) & BIT_MASK2(start, end)) >> (

    uint32_t mask = GET_ADDR_BITS(inst, 7, 0);
    uint32_t mask2 = GET_ADDR_BITS(inst, 15, 12);
    uint32_t mask3 = GET_ADDR_BITS(inst, 32, 25);

    if (mask == 0b0110111)
    {
        return LUI;
    }
    else if (mask == 0b0010111)
    {
        return AUIPC;
    }
    else if (mask == 0b1101111)
    {
        return JAL;
    }
    else if (mask == 0b1100111)
    {
        return JALR;
    }
    else if (mask == 0b1100011)
    {
        if (mask2 == 0b000)
        {
            return BEQ;
        }
        else if (mask2 == 0b001)
        {
            return BNE;
        }
        else if (mask2 == 0b100)
        {
            return BLT;
        }
        else if (mask2 == 0b101)
        {
            return BGE;
        }
        else if (mask2 == 0b110)
        {
            return BLTU;
        }
        else if (mask2 == 0b111)
        {
            return BGEU;
        }
```

```
    }
    else if (mask == 0b0000011)
    {
        if (mask2 == 0b000)
        {
            return LB;
        }
        else if (mask2 == 0b001)
        {
            return LH;
        }
        else if (mask2 == 0b010)
        {
            return LW;
        }
        else if (mask2 == 0b100)
        {
            return LBU;
        }
        else if (mask2 == 0b101)
        {
            return LHU;
        }
    }
    else if (mask == 0b0100011)
    {
        if (mask2 == 0b000)
        {
            return SB;
        }
        else if (mask2 == 0b001)
        {
            return SH;
        }
        else if (mask2 == 0b010)
        {
            return SW;
        }
    }
    else if (mask == 0b0010011)
    {
        if (mask2 == 000)
        {
            return ADDI;
        }
        else if (mask2 == 0b010)
        {
            return SLTI;
        }
        else if (mask2 == 0b011)
        {
            return SLTIU;
        }
```

```
107             else if (mask2 == 0b100)
108             {
109                 return XORI;
110             }
111             else if (mask2 == 0b110)
112             {
113                 return ORI;
114             }
115             else if (mask2 == 0b111)
116             {
117                 return ANDI;
118             }
119             else if (mask2 == 0b001)
120             {
121                 return SLLI;
122             }
123             else if (mask2 == 0b101)
124             {
125                 if (mask3 == 0b0000000)
126                 {
127                     return SRLI;
128                 }
129                 else if (mask3 == 0b0100000)
130                 {
131                     return SRAI;
132                 }
133             }
134         }
135         else if (mask == 0b0110011)
136         {
137             if (mask2 == 0b000)
138             {
139                 if (mask3 == 0b0000000)
140                 {
141                     return ADD;
142                 }
143                 else if (mask3 == 0b0100000)
144                 {
145                     return SUB;
146                 }
147             }
148             else if (mask2 == 0b001)
149             {
150                 return SLL;
151             }
152             else if (mask2 == 0b010)
153             {
154                 return SLT;
155             }
156             else if (mask2 == 0b011)
157             {
158                 return SLTU;
159             }
```

```
160             else if (mask2 == 0b100)
161             {
162                 return XOR;
163             }
164             else if (mask2 == 0b101)
165             {
166                 if (mask3 == 0b0000000)
167                 {
168                     return SRL;
169                 }
170                 else if (mask3 == 0b0100000)
171                 {
172                     return SRA;
173                 }
174             }
175             else if (mask2 == 0b110)
176             {
177                 return OR;
178             }
179             else if (mask2 == 0b111)
180             {
181                 return AND;
182             }
183         }
184         else if (mask == 0b0001111)
185         {
186             if (mask2 == 0b000)
187             {
188                 return FENCE;
189             }
190             else if (mask2 == 0b001)
191             {
192                 return FENCEI;
193             }
194         }
195         else if (mask == 0b1110011)
196         {
197             if (mask2 == 0b000)
198             {
199                 if (mask3 == 0b0000000)
200                 {
201                     return ECALL;
202                 }
203                 else if (mask3 == 0b0000001)
204                 {
205                     return EBREAK;
206                 }
207             }
208             else if (mask2 == 0b001)
209             {
210                 return CSRRW;
211             }
212             else if (mask2 == 0b010)
213             {
```

```
213             {
214                 return CSRRS;
215             }
216             else if (mask2 == 0b011)
217             {
218                 return CSRRC;
219             }
220             else if (mask2 == 0b101)
221             {
222                 return CSRRWI;
223             }
224             else if (mask2 == 0b110)
225             {
226                 return CSRRSI;
227             }
228             else if (mask2 == 0b111)
229             {
230                 return CSRRCI;
231             }
232         }
233         else
234         {
235             return UNDEFINED;
236         }
237     }
```

Execution Result

```
Testing get_bit()

get_bit(0x0000004e,0): 0x00000000, correct
get_bit(0x0000004e,1): 0x00000001, correct
get_bit(0x0000004e,5): 0x00000000, correct
get_bit(0x0000001b,3): 0x00000001, correct
get_bit(0x0000001b,2): 0x00000000, correct
get_bit(0x0000001b,9): 0x00000000, correct

Testing set_bit()

set_bit(0x0000004e,2,0): 0x0000004a, correct
set_bit(0x0000006d,0,0): 0x0000006c, correct
set_bit(0x0000004e,2,1): 0x0000004e, correct
set_bit(0x0000006d,0,1): 0x0000006d, correct
set_bit(0x0000004e,9,0): 0x0000004e, correct
set_bit(0x0000006d,4,0): 0x0000006d, correct
set_bit(0x0000004e,9,1): 0x0000024e, correct
set_bit(0x0000006d,7,1): 0x000000ed, correct

Testing flip_bit()

flip_bit(0x0000004e,0): 0x0000004f, correct
flip_bit(0x0000004e,1): 0x0000004c, correct
flip_bit(0x0000004e,2): 0x0000004a, correct
flip_bit(0x0000004e,5): 0x0000006e, correct
flip_bit(0x0000004e,9): 0x0000024e, correct


Testing decode_riscv_inst()

Your decode function is correct.
```

HW 2-2 Linear feedback shift register

Code

Please copy and paste your code for the following function in this section.

- lfsr_calculate()

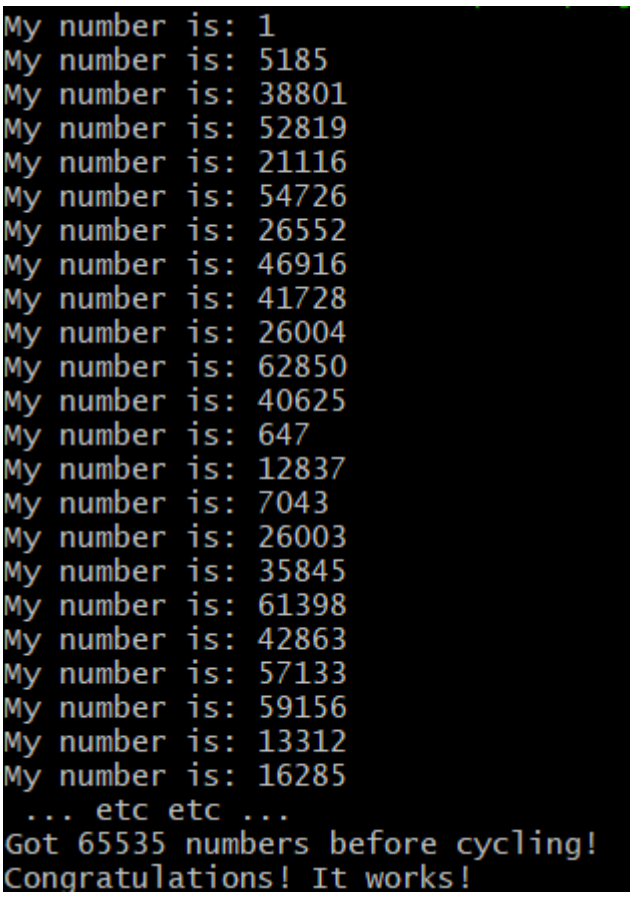
Remember to write clear comment in your code to explain your implementation in more details.

lfsr_calculate()

這個function主要是16bit的亂數產生器，透過不斷從最高位元更新新 `bit` 使其跑出15種不同的排列組合，而產生新 `bi` t的方法為：index0(數值)先跟index2做 ^ ，再和index3做 ^ ，最後再和index5做 ^ ，其中每次都將指定index右移到最低位再去做 ^ ，每次完成 ^ 需先 `&1` 再做下次計算，將其餘不相干的 `bit` 化為0，僅保留欲更新的 `bit` 。完成後將 `reg` 往右為移1位，得到的新 `bit` 左移15位更新成為 `reg` 的最高位 `bit` 。

```
1  void lfsr_calculate(uint16_t *reg) {
2      unsigned bit = ((((*reg >> 0) ^ (*reg >> 2) & 1) ^ (*reg >> 3) & 1) ^ (*reg
3      *reg = (*reg >> 1) | (bit << 15);
4  }
```

Execution Result



HW 2-3 Memory Management

Explanation

> Please explain why bad_vector_new() and also_bad_vector_new() are two examples of bad ways to initialize.

bad_vector_new()

因為 `vector_t *retval, v` 這段程式碼使得 `v` 宣告時是靜態陣列，而 `retval` 指向 `v` ，之後要 `free` 出空間時，會因為 `v` 並非靜態陣列而出現問題。

```
/* Bad example of how to create a new vector */
vector_t *bad_vector_new() {
        /* Create the vector and a pointer to it */
        vector_t *retval, v;
        retval = &v;

        /* Initialize attributes */
        retval->size = 1;
        retval->data = malloc(sizeof(int));
        if (retval->data == NULL) {
                allocation_failed();
        }

        retval->data[0] = 0;
        return retval;
}
```

also_bad_vector_new()

因為 `vector_t v` 宣告 `v` 為靜態陣列，也並沒有用指標指著，所以當該function結束，靜態陣列會消失而無法 `return` 。

```
/* Another suboptimal way of creating a vector */
vector_t also_bad_vector_new() {
        /* Create the vector */
        vector_t v;

        /* Initialize attributes */
        v.size = 1;
        v.data = malloc(sizeof(int));
        if (v.data == NULL) {
                allocation_failed();
        }
        v.data[0] = 0;
        return v;
}
```

Code

Please copy and paste your code for the following 4 functions in this section.

- vector_new()
- vector_get()
- vector_delete()
- vector_set()

Remember to write clear comment in your code to explain your implementation in more details.

vector_new()

創建一個指向 `vector_t` 型態的指標 `retval` ，指向 `vector_t` 型態宣告的動態陣列，其中 `vector_t` 有兩個值須賦予，分別是int型態的 `size` 和存資料的動態陣列 `data` 。

```
/* Create a new vector with a size (length) of 1
   and set its single component to zero... the
   RIGHT WAY */
vector_t *vector_new() {
        /* Declare what this function will return */
        vector_t *retval;
        //retval指向一個vector型態的動態陣列，vector內含有size和data，data後面需再指向
        /* First, we need to allocate memory on the heap for the struct */
        retval = malloc(sizeof(vector_t));

        /* Check our return value to make sure we got memory */
        if (retval==NULL) {
                allocation_failed();
        }

        /* Now we need to initialize our data.
           Since retval->data should be able to dynamically grow,
           what do you need to do? */
        retval->size = 0;
        retval->data = malloc(sizeof(int)*1);

        /* Check the data attribute of our vector to make sure we got memory */
        if (retval->data== NULL) {
                free(retval);
         allocation_failed();
        }

        /* Complete the initialization by setting the single component to zero *
        *(retval->data) = 0;
        /* and return... */
        return retval;
}
```

vector_get()

若 `loc` 在該vector的 `data` 範圍內，則使用 `v->data[loc]` 取值。

```
/* Return the value at the specified location/component "loc" of the vector */
int vector_get(vector_t *v, size_t loc) {

        /* If we are passed a NULL pointer for our vector, complain about it and
        if(v == NULL) {
                fprintf(stderr, "vector_get: passed a NULL vector.\n");
         abort();
        }

        /* If the requested location is higher than we have allocated, return 0.
         * Otherwise, return what is in the passed location.
         */
        if (loc < v->size) {
                return v->data[loc];
        } else {
                return 0;
        }
}
```

vector_delete()

將 `v` 刪除釋放出空間，需先釋放該vector內 `data` 的動態陣列，再釋放 `v` 本身的動態陣列。

```
/* Free up the memory allocated for the passed vector.
   Remember, you need to free up ALL the memory that was allocated. */
void vector_delete(vector_t *v) {
    free(v->data)
    free(v);
}
```

vector_set()

若 `loc` 超過vector大小，則 `realloc` 重新分配 `data` 大小，否則直接取代該 `loc` 所在位置的值。

```
/* Set a value in the vector. If the extra memory allocation fails, call
   allocation_failed(). */
void vector_set(vector_t *v, size_t loc, int value) {
        /* What do you need to do if the location is greater than the size we ha
         * allocated?  Remember that unset locations should contain a value of 0
         */

        size_t newsize = loc + 1;
        size_t orisize = v->size;
        if (orisize < newsize)
        {
                //如果原size小於新size，則將原data擴建
                int* newdata = realloc(v->data, sizeof(int) * newsize);
                //在新建的空間放入0
                for (int i = orisize; i < newsize; i++)
                {
                        newdata[i] = 0;
                }
                newdata[loc] = value;
                int s = newdata[loc];
                v->data = newdata;
                v->size = newsize;
        }
        else
        {
                v->data[loc] = value;
        }
}
```

The result of using Valgrind to test memory leak.

```
Calling vector_new()
Calling vector_delete()
vector_new() again
These should all return 0 (vector_get()): 0 0 0
Doing a bunch of vector_set()s
These should be equal:
98 = 98
15 = 15
65 = 65
-123 = -123
21 = 21
43 = 43
0 = 0
0 = 0
0 = 0
3 = 3
Test complete.
==662==
==662== HEAP SUMMARY:
==662==     in use at exit: 0 bytes in 0 blocks
==662==   total heap usage: 10 allocs, 10 frees, 3,284 bytes allocated
==662==
==662== All heap blocks were freed -- no leaks are possible
==662==
==662== For lists of detected and suppressed errors, rerun with: -s
==662== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

HW 2-4 AI model Statistics

Code

> Please copy and paste your code for parse_model.py.
> Remember to write clear comment in your code to explain your implementation in more details.

計算所用到之乘法數量：
(kernel width * kernel hight) * (kernel M * kernel C) * (input H * input W)
其中 `inputsize` 陣列是從 `value_info_nlist` 取得，紀錄各個conv內的input長和寬(iH,iW)的乘積，再去和 `input_nlist` 對應conv的 `size` (kw,kh,kM,kC)做相乘得到每層conv的乘法數量。

```python
## parse_model.py

import onnx

onnx_model = onnx.load('./lenet.onnx')

## need to run shape inference in order to get a full value_info list
onnx_model = onnx.shape_inference.infer_shapes(onnx_model)

## List all tensor names in the raph
input_nlist = [k.name for k in onnx_model.graph.input]##將k.name放入nlist
initializer_nlist = [k.name for k in onnx_model.graph.initializer]
value_info_nlist = [k.name for k in onnx_model.graph.value_info]#maxpool

print('\ninput list: {}'.format(input_nlist))
print('\ninitializer list: {}'.format(initializer_nlist))
print('\nvalue_info list: {}'.format(value_info_nlist))

## a simple function to calculate the tensor size and extract dimension informat
def get_size(shape):
    dims = []
    ndim = len(shape.dim)
    size = 1
    for i in range(ndim):
        size = size * shape.dim[i].dim_value
        dims.append(shape.dim[i].dim_value)
    return dims, size

count = []
inputsize = []
ind=0
## find all `Conv` operators and print its input information
for i in onnx_model.graph.node:
    if (i.op_type == 'Conv'):
        print('\n-- Conv "{}" --'.format(i.name))
        for j in i.input:
            if j in input_nlist:
                idx = input_nlist.index(j)
                (dims, size) = get_size(onnx_model.graph.input[idx].type.tensor_
                print('input {} has {} elements dims = {}'.format(j, size, dims)
                #該conv所用到之乘法數量：(kernel width * kernel hight) * (kernel M
                if(size==784 or size ==800):
                    if (size==800):
                        count.append(size*28*28)
                else:
                    count.append(size*inputsize[ind])
                    ind = ind+1
            elif j in initializer_nlist:
                idx = initializer_nlist.index(j)
                (dims, size) = get_size(onnx_model.graph.initializer[idx].type.t
                print('input {} has {} elements dims = {}'.format(j, size, dims)
            elif j in value_info_nlist:
                idx = value_info_nlist.index(j)
```

```
                (dims, size) = get_size(onnx_model.graph.value_info[idx].type.te
                print('input {} has {} elements dims = {}'.format(j, size, dims)
                inputsize.append(int(size/dims[1]))
print('multiplication operations')
for i in range(4):
    print('conv{}:{}'.format(i,count[i]))
```

Execution Result

```
multiplication operations
conv0:627200
conv1:10035200
conv2:157351936
conv3:10240
```

Bonus

Please document your bonus home in this section or attach links to related Gitlab code or documents

Others

- If you have any comment or recommendation to this lab, you can write it down here to tell us.