HW2

b09902004 資工二 郭懷元

2.8

```
addi x30, x10, 8 ⟶ x30 = A+1
addi x31, x10, 0 ⟶ x31 = A
sd x31, 0(x30) ⟶ *(x30) = x31 ⟶ *(A+1) = A ⟶ A[1] = A
ld x30, 0(x30) ⟶ x30 = *(x30) ⟶ x30 = *(A+1) = A
add x5, x30, x31 ⟶ f = A + A
```

```
A[1] = A;
f = A[1] + A;
```

2.9

instruction	opcode	rs1	rd	rs2	imm	funct3	funct7
`addi x30, x10, 8`	`010011`	`01010`	`11110`		`0x008`	`000`	
`addi x31, x10, 0`	`010011`	`01010`	`11111`		`0x000`	`000`	
`sd x31, 0(x30)`	`100011`	`11110`		`11111`	`0x000`	`011`	
`ld x30, 0(x30)`	`000011`	`11110`	`11110`		`0x000`	`011`	
`add x5, x30, x31`	`110011`	`11110`	`00110`	`11111`		`000`	`000`

2.16

2.16.1

`funct7` , `funct3` , `opcode` : These bit fields might increase in size to accommodate the four times as many instructions.

`rs2` , `rs1` , `rd` : These bit fields should increase from 5 bits to 7 bits for the 128 registers.

2.16.2

`funct3` , `opcode` : These bit fields might increase in size to accommodate the four times as many instructions.

`rs1` , `rd` : These bit fields should increase from 5 bits to 7 bits for the 128 registers.

`imm` : This field doesn't need to change, because neither the number of registers or instructions have to do with `imm` .

2.16.3

Decrease in size: Because there are more registers and more instructions, some old instructions can now be combined into just a single instruction.

Increase in size: Because instructions now takes up more bits, for simple tasks that doesn't use many registers, the extra bits are wasted and take up unnecessary spaces.

Matrix Multiplication Report

> Refs:
>
> https://en.wikipedia.org/wiki/Loop_nest_optimization
> https://github.com/flame/how-to-optimize-gemm/wiki

Result

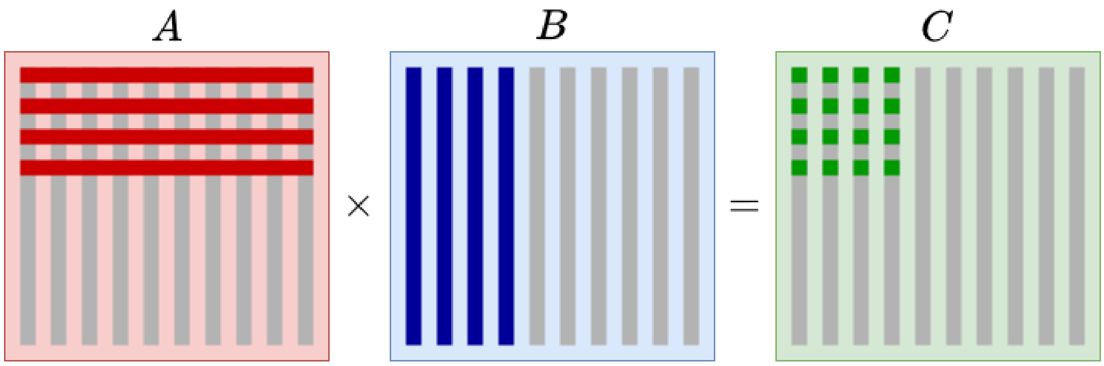
Cycle count: `5630353`

問題 輸出 終端機 偵錯主控台 連接埠

```
root@b06803fdad65:~/Problems/matrix# make run
riscv64-unknown-elf-gcc -O3 -o matrix matrix.c matrix.s
root@b06803fdad65:~/Problems/matrix# make test
spike pk ./matrix
bbl loader
Took 5630353 cycles
root@b06803fdad65:~/Problems/matrix# 
```

Strategy

- Dot 4 rows of A and 4 columns of B at a time, so that when a value from A or B is read from memory to register, it can be used multiple times.



- Use pointers and indirect addressing to access A, B, and C.
- Only do the modulo operation before storing to C, because we are using `unsigned short`, and overflows in `unsigned` variables are essentially modulo.
- In the `for` loop of `k`, do two iterations at once, reducing call for branching operations.
- When inside the `for` loop of `i`, put `i` from register to stack, and use this register for as temporary register for calculation.
- I tried memory blocking to keep data in L1 and L2 cache, but somehow it didn't work, and the performance is even worse.
- Registers are barely enough. Unrolling some loops could avoid the use of `gp`, `tp`, and `ra`.

reg	usage	reg	usage
ra	k	s7	B[k][j+1]
t0	i / tmp	s8	B[k][j+2]
t1	j	s9	B[k][j+3]
t2	acc20	s10	acc31
t3	acc21	s11	acc32
t4	acc22	a0	A
t5	acc23	a1	B
t6	acc30	a2	C
s0	acc00	a3	acc33
s1	acc01	a4	acc10
s2	acc02	a5	acc11
s3	acc03	a6	acc12
s4	A[i][k]	a7	acc13
s5	A[i+1][k]	gp	A[i+2][k]
s6	B[k][j]	tp	A[i+3][k]