ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KĨ THUẬT MÁY TÍNH

KIẾN TRÚC MÁY TÍNH (CO2008)

BÀI TẬP LỚN
ĐỀ: NHÂN HÀI SỐ NGUYÊN 32 BITS

GVHD: Nguyễn Xuân Minh
Sinh viên: Lê Nguyễn Hải Đăng - MSSV:2113176
Trần Minh Hiếu - MSSV:2113363

Tp. Hồ Chí Minh, tháng 12, 2022

Contents

1 Danh sách thành viên và phân công công việc

No.	Họ và Tên	Mssv	Công Việc	Thực hiện
1	Lê Nguyễn Hải Đăng	2113176	Viết báo cáo, Thực thi code	100%
2	Trần Minh Hiếu	2113363	Viết báo cáo, Thực thi code	100%

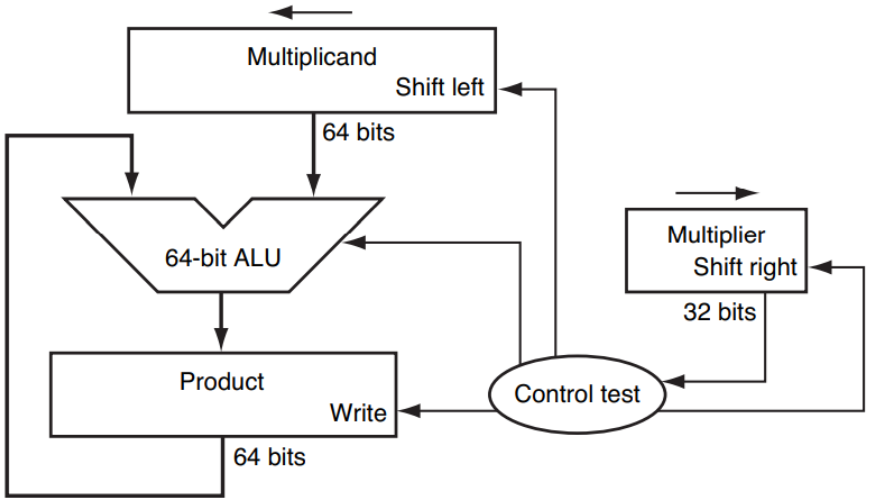
2 Yêu cầu

- Sử dụng tập lệnh MIPS viết chương trình thực hiện phép nhân số nguyên 32 bit
- Tính thời gian chạy của chương trình: sử dụng tần số 1 GHz để tính trong bài báo cáo này
- Code Style phải rõ ràng, có comment.

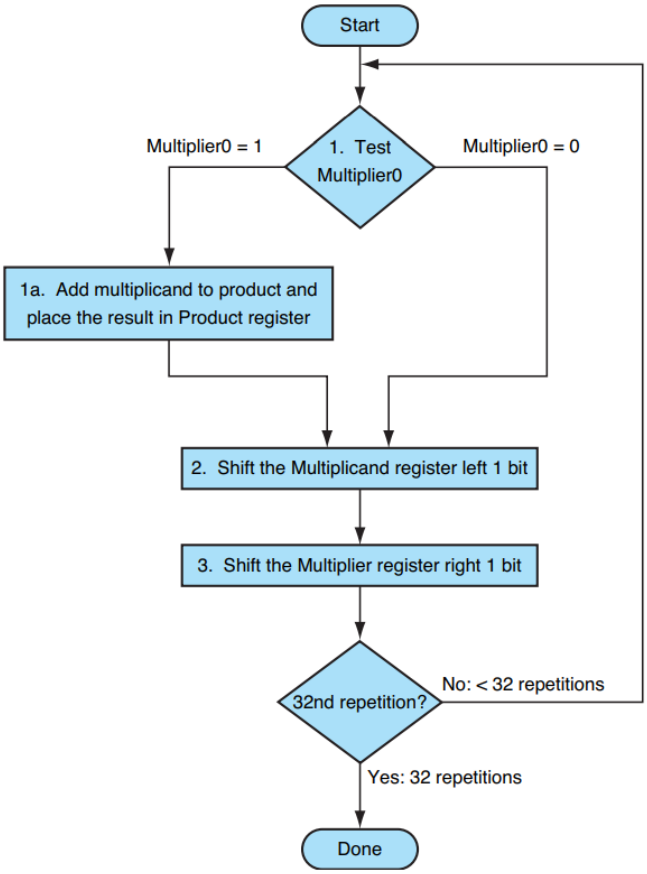
  Phải có gọi hàm. Truyền tham số và kết quả khi gọi hàm theo quy ước của thanh ghi. In kết quả ra màn hình để kiểm tra.
- Báo cáo: nộp 2 file: Diễn giải (file.pdf) và Mã nguồn (file.asm) và file input đầu vào INT.BIN

3 Giải thuật nhân 2 số nguyên 32 bits

Trong kiến trúc của bộ nhân, số nhân được chứa trong thanh ghi 32 bit. Thanh ghi tích sẽ có kích thước 64 bit và được khởi tạo bằng 0. Theo giải thuật nhân bằng giấy và bút, số bị nhân sẽ bị dịch sáng trái ở mỗi bước tính toán (tương ứng với mỗi ký số của số nhân). Giá trị này sẽ được công vào các giá trị tích trung gian (intermmediate products) nếu ký số đang xét ở số nhân là 1.Bởi vì số nhân có kích thước là 32 bit nên số bị nhân sẽ cần được dịch sang trái 32 lân. Do đó, cần dùng thanh ghi 64 bit để chứa số bị nhân để đảm bảo rằng việc dịch trái không gây ra mất mát bất kỳ bit nào của số bị nhân. Ở thời điểm ban đầu, thanh ghi 64 bit này sẽ chứa 32 bit của số bị nhân ở 32 bit thấp và 32 bit cao được gán bằng 0.

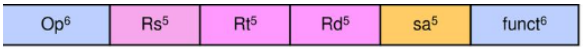


Trong kiến trúc này, ở mỗi bước tính toán thanh ghi chứa số nhân sẽ được dịch sang phải 1 bit và bit cuối cùng sẽ được xử lý. Gọi $multiplier_0$ là bit cuối cùng của thanh ghi số nhân, nếu giá trị của $multiplier_0$ là 0 thì dịch trái chứa số bị nhân và dịch phải thanh ghi chứa số nhân, ngược lại thực hiện việc cộng dồn giá trị hiện tại trong thanh ghi số bị nhân vào thanh ghi tích rồi thực hiện dịch trái thanh ghi số bị nhân 1 bit vào dịch phải thanh ghi số nhân 1 bit.Giá trị tại thời điểm này của thanh ghi tích chính là kết quả của phép nhân.



4 Thống kê số lệnh và các loại lệnh của mỗi nhóm

4.1 R-Type



- Op: mã phép toán (Opcode): Trường này sẽ cho máy biết lệnh này là lệnh nào. Trong trường hợp R-formatthì các lệnh đều chung opcode là 0.
- Funct: Vì các lệnh R-format đều có chung opcode bằng 0 nên ta thêm trường này để cho máy biết cần thực hiện lệnh nào.

- Rs, Rt: source register và destination register, hai thanh ghi cần thực hiện tính toán.
- Rd: register destination, thanh ghi lưu kết quả của lệnh.
- Shamt: shift amount, số bit cần dịch trong lệnh dịch trái và dịch phải.

Các lệnh R-Type đã được sử dụng :

STT	Tên Lệnh	Chức năng
1	sll	R[rd]=R[rt]<<shamt
2	srl	R[rd]=R[rt]>>shamt
3	move	R[rd]=R[rs]
4	addu	R[rd]=R[rs]+R[rt]
5	sltu	R[rd]=(R[rs]<R[rt])?1:0
6	jr	PC=R[rs]
7	xor	R[rd]=R[rs] xor R[rt]

4.2 I-Type

- Op: mã phép toán (Opcode): Trường này sẽ cho máy biết lệnh này là lệnh nào. Vì I-format không dùng chung opcode như các lệnh R-format.
- Rs, Rt: source register và destination register.
- Immediate: Một giá trị hằng số mà lệnh sử dụng.

Các lệnh I-Type đã được sử dụng :

STT	Tên Lệnh	Chức năng
1	lw	R[rt]=M[R[rs]+SignExtImm]
2	beq	if(R[rs]==R[rt])PC=PC+4+BranchAddr
3	bgt	if(R[rs]>R[rt])PC=Label
4	sw	M[R[rs]+SignExtImm]=R[rt]
5	addi	R[rd]=R[rs]+SignExtImm

4.3 J-Type

- Op: mã phép toán (Opcode): Trường này sẽ cho máy biết lệnh này là lệnh nào.
- Target address: Địa chỉ rút gọn của lệnh cần nhảy đến, địa chỉ gốc có 32 bit, ta rút gọn 6 bit như sau:

  Xóa 2 bit thấp nhất của địa chỉ. Vì địa chỉ của các lệnh trong MIPS luôn chia hết cho 4 nên 2 bit thấp nhất luôn bằng 0.Và 4 bit cao nhất xem như bằng với 4 bit cao nhất của lệnh hiện tại.

Các lệnh J-format đã sử dụng:

STT	Tên Lệnh	Chức Năng
1	j	PC=JumpAddr
2	jal	R[31]=PC+8;PC=JumpAddr

5 Hiện thực nhân hai số nguyên 32 bit trên MIPS

5.1 Hiện thực code MIPS

5.2 Kiểm tra testcase

Trong đó:

- Instructions so far là tổng số câu lệnh mà chương trình đã thực thi.
- R-type là tổng số câu lệnh loại R mà chương trình đã thực thi.
- I-type là tổng số câu lệnh loại I mà chương trình đã thực thi.
- J-type là tổng số câu lệnh loại J mà chương trình đã thực thi.

5.2.1 Testcase 1:

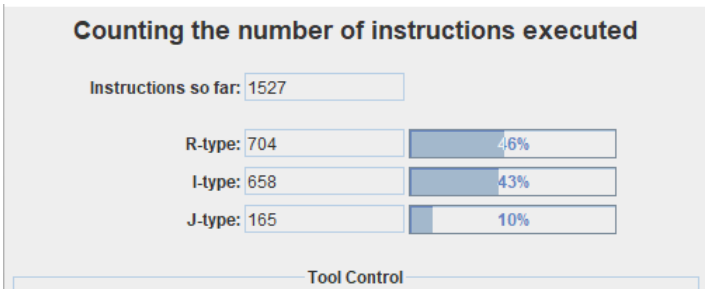
- Số bị nhân : 12
- Số nhân : 4
- Kết quả : 48

Kết quả testcase 1 hiển thị trong Mars Mips:

```
Multiplicand_2:                                 00000000000000000000000000001100
Multiplier_2:                                   00000000000000000000000000000100
Multiplicand:_10:                               12
Multiplier:_10:                                 4
Ket qua la (2):  00000000000000000000000000000000 00000000000000000000000000110000
High 32_bit:                                    0
Low 32_bit:                                     48

-- program is finished running --
```

Số lệnh đo được trong quá trình thực thi:



Thời gian thực thi tính được là:

$$Time = \frac{InstructionCount * CPI}{ClockRate} = \frac{1527 * 1}{10^9} = 1527ns$$

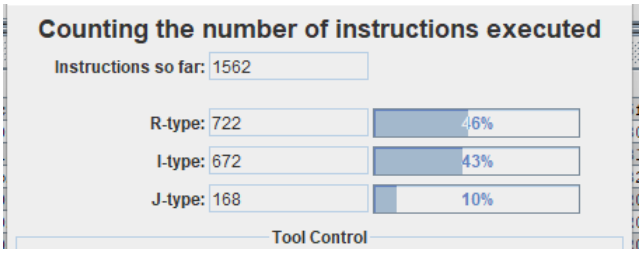
5.2.2 Testcase 2:

- Số bị nhân : 12
- Số nhân : -5
- Kết quả : -60

Kết quả testcase 2 hiển thị trong Mars Mips:

```
Multiplicand_2:                          00000000000000000000000000001100
Multiplier_2:                            11111111111111111111111111111011
Multiplicand:_10:                        12
Multiplier:_10:                          -5
Ket qua la (2):  00000000000000000000000000000000 11111111111111111111111111000100
High 32_bit:                             0
Low 32_bit:                              -60
```

Số lệnh đo được trong quá trình thực thi:



Thời gian thực thi tính được là:

$$Time = \frac{InstructionCount * CPI}{ClockRate} = \frac{1562 * 1}{10^9} = 1562ns$$

5.2.3 Testcase 3:

- Số bị nhân : 400,000
- Số nhân : 400,000
- Kết quả : 1,600,000,000

Kết quả testcase 3 hiển thị trong Mars Mips:

```
Multiplicand_2:                          00000000000001100001101010000000
Multiplier_2:                            00000000000001100001101010000000
Multiplicand:_10:                        400000
Multiplier:_10:                          400000
Ket qua la (2):  00000000000000000000000100101 01000000101111100100000000000000
High 32_bit:                             37
Low 32_bit:                              1086210048

-- program is finished running --
```

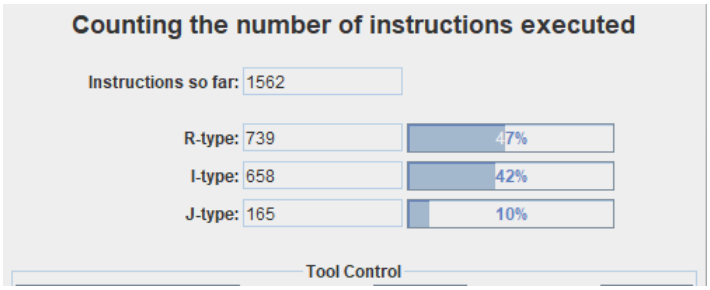
Trong trường hợp kết quả chứa trong hai thanh ghi ta áp dụng công thức dưới đây để tính kết quả:

$$Result = RegMBS * 2^{32} + RegLSB$$

Vậy kết quả chứa hai thanh ghi trên sẽ là:

$$37 * 2^{32} + 1086210048 = 1600000000$$

Số lệnh đo được trong quá trình thực thi:



Thời gian thực thi tính được là:

$$Time = \frac{InstructionCount * CPI}{ClockRate} = \frac{1562 * 1}{10^9} = 1562ns$$

References

[1] David Patterson and John Hennessy. Computer Organization and Design: the Hardware Software Interface, 5 th edition. 2014.

[2] MIPS Technology, Inc. MIP S32TM Architecture For Programmers Volume II: The MIP S32TM Instruction Set. 2001-2003.