

## Solution to Practice Final Examination

## 1 Multi-Cycle Implementation

1. The control signals for the given sequence of instructions.

Cycle 1: Fetch `rmmov %rax, 0x100(%rcx)`

<i>nRegid</i>	<i>nValC</i>	<i>regA</i>	<i>regB</i>	<i>regE</i>	<i>regM</i>	<i>aluA</i>	<i>aluB</i>
1	1	XX	XX	00	00	XX	X
<i>aluF</i>	<i>setCnd</i>	<i>mAddr</i>	<i>mData</i>	<i>mRd</i>	<i>mWrt</i>	<i>newPC</i>	<i>vPWrt</i>
XX	0	X	X	X	0	XX	1
<i>irWrt</i>	<i>vAWrt</i>	<i>vBWrt</i>	<i>vEWrt</i>	<i>vMWrt</i>	<i>pcWrt</i>		
1	0	0	0	0	0		

Cycle 2: Decode

<i>nRegid</i>	<i>nValC</i>	<i>regA</i>	<i>regB</i>	<i>regE</i>	<i>regM</i>	<i>aluA</i>	<i>aluB</i>
X	X	10	10	00	00	XX	X
<i>aluF</i>	<i>setCnd</i>	<i>mAddr</i>	<i>mData</i>	<i>mRd</i>	<i>mWrt</i>	<i>newPC</i>	<i>vPWrt</i>
XX	0	X	X	X	0	XX	0
<i>irWrt</i>	<i>vAWrt</i>	<i>vBWrt</i>	<i>vEWrt</i>	<i>vMWrt</i>	<i>pcWrt</i>		
0	1	1	0	0	0		

Cycle 3: Execute

<i>nRegid</i>	<i>nValC</i>	<i>regA</i>	<i>regB</i>	<i>regE</i>	<i>regM</i>	<i>aluA</i>	<i>aluB</i>
X	X	XX	XX	00	00	10	1
<i>aluF</i>	<i>setCnd</i>	<i>mAddr</i>	<i>mData</i>	<i>mRd</i>	<i>mWrt</i>	<i>newPC</i>	<i>vPWrt</i>
00	0	X	X	X	0	XX	0
<i>irWrt</i>	<i>vAWrt</i>	<i>vBWrt</i>	<i>vEWrt</i>	<i>vMWrt</i>	<i>pcWrt</i>		
0	0	0	1	0	0		

Cycle 4: Memory

<i>nRegid</i>	<i>nValC</i>	<i>regA</i>	<i>regB</i>	<i>regE</i>	<i>regM</i>	<i>aluA</i>	<i>aluB</i>
X	X	XX	XX	00	00	XX	X
<i>aluF</i>	<i>setCnd</i>	<i>mAddr</i>	<i>mData</i>	<i>mRd</i>	<i>mWrt</i>	<i>newPC</i>	<i>vPWrt</i>
XX	0	0	0	0	1	XX	0
<i>irWrt</i>	<i>vAWrt</i>	<i>vBWrt</i>	<i>vEWrt</i>	<i>vMWrt</i>	<i>pcWrt</i>		
0	0	0	0	0	0		

Cycle 5: PC Update

<i>nRegid</i>	<i>nValC</i>	<i>regA</i>	<i>regB</i>	<i>regE</i>	<i>regM</i>	<i>aluA</i>	<i>aluB</i>
X	X	XX	XX	00	00	XX	X
<i>aluF</i>	<i>setCnd</i>	<i>mAddr</i>	<i>mData</i>	<i>mRd</i>	<i>mWrt</i>	<i>newPC</i>	<i>vPWrt</i>
XX	0	X	X	X	0	00	0
<i>irWrt</i>	<i>vAWrt</i>	<i>vBWrt</i>	<i>vEWrt</i>	<i>vMWrt</i>	<i>pcWrt</i>		
0	0	0	0	0	1		

Cycle 6: Fetch `add %rcx, %rdx`

<i>nRegid</i>	<i>nValC</i>	<i>regA</i>	<i>regB</i>	<i>regE</i>	<i>regM</i>	<i>aluA</i>	<i>aluB</i>
1	0	XX	XX	00	00	XX	X
<i>aluF</i>	<i>setCnd</i>	<i>mAddr</i>	<i>mData</i>	<i>mRd</i>	<i>mWrt</i>	<i>newPC</i>	<i>vPWrt</i>
XX	0	X	X	X	0	XX	1
<i>irWrt</i>	<i>vAWrt</i>	<i>vBWrt</i>	<i>vEWrt</i>	<i>vMWrt</i>	<i>pcWrt</i>		
1	0	0	0	0	0		

Cycle 7: Decode

<i>nRegid</i>	<i>nValC</i>	<i>regA</i>	<i>regB</i>	<i>regE</i>	<i>regM</i>	<i>aluA</i>	<i>aluB</i>
X	X	10	10	00	00	XX	X
<i>aluF</i>	<i>setCnd</i>	<i>mAddr</i>	<i>mData</i>	<i>mRd</i>	<i>mWrt</i>	<i>newPC</i>	<i>vPWrt</i>
XX	0	X	X	X	0	XX	0
<i>irWrt</i>	<i>vAWrt</i>	<i>vBWrt</i>	<i>vEWrt</i>	<i>vMWrt</i>	<i>pcWrt</i>		
0	1	1	0	0	0		

Cycle 8: Execute

<i>nRegid</i>	<i>nValC</i>	<i>regA</i>	<i>regB</i>	<i>regE</i>	<i>regM</i>	<i>aluA</i>	<i>aluB</i>
X	X	XX	XX	00	00	11	1
<i>aluF</i>	<i>setCnd</i>	<i>mAddr</i>	<i>mData</i>	<i>mRd</i>	<i>mWrt</i>	<i>newPC</i>	<i>vPWrt</i>
00	1	X	X	X	0	XX	0
<i>irWrt</i>	<i>vAWrt</i>	<i>vBWrt</i>	<i>vEWrt</i>	<i>vMWrt</i>	<i>pcWrt</i>		
0	0	0	1	0	0		

Cycle 9: Write Back

<i>nRegid</i>	<i>nValC</i>	<i>regA</i>	<i>regB</i>	<i>regE</i>	<i>regM</i>	<i>aluA</i>	<i>aluB</i>
X	X	XX	XX	10	00	XX	X
<i>aluF</i>	<i>setCnd</i>	<i>mAddr</i>	<i>mData</i>	<i>mRd</i>	<i>mWrt</i>	<i>newPC</i>	<i>vPWrt</i>
XX	0	X	X	X	0	XX	0
<i>irWrt</i>	<i>vAWrt</i>	<i>vBWrt</i>	<i>vEWrt</i>	<i>vMWrt</i>	<i>pcWrt</i>		
0	0	0	0	0	0		

Cycle 10: PC Update

<i>nRegid</i>	<i>nValC</i>	<i>regA</i>	<i>regB</i>	<i>regE</i>	<i>regM</i>	<i>aluA</i>	<i>aluB</i>
X	X	XX	XX	00	00	XX	X
<i>aluF</i>	<i>setCnd</i>	<i>mAddr</i>	<i>mData</i>	<i>mRd</i>	<i>mWrt</i>	<i>newPC</i>	<i>vPWrt</i>
XX	0	X	X	X	0	00	0
<i>irWrt</i>	<i>vAWrt</i>	<i>vBWrt</i>	<i>vEWrt</i>	<i>vMWrt</i>	<i>pcWrt</i>		
0	0	0	0	0	1		

Cycle 11: Fetch `push %rdx`

<i>nRegid</i>	<i>nValC</i>	<i>regA</i>	<i>regB</i>	<i>regE</i>	<i>regM</i>	<i>aluA</i>	<i>aluB</i>
1	0	XX	XX	00	00	XX	X
<i>aluF</i>	<i>setCnd</i>	<i>mAddr</i>	<i>mData</i>	<i>mRd</i>	<i>mWrt</i>	<i>newPC</i>	<i>vPWrt</i>
XX	0	X	X	X	0	XX	1
<i>irWrt</i>	<i>vAWrt</i>	<i>vBWrt</i>	<i>vEWrt</i>	<i>vMWrt</i>	<i>pcWrt</i>		
1	0	0	0	0	0		

## Cycle 12: Decode

<i>nRegid</i>	<i>nValC</i>	<i>regA</i>	<i>regB</i>	<i>regE</i>	<i>regM</i>	<i>aluA</i>	<i>aluB</i>
X	X	10	01	00	00	XX	X
<i>aluF</i>	<i>setCnd</i>	<i>mAddr</i>	<i>mData</i>	<i>mRd</i>	<i>mWrt</i>	<i>newPC</i>	<i>vPWrt</i>
XX	0	X	X	X	0	XX	0
<i>irWrt</i>	<i>vAWrt</i>	<i>vBWrt</i>	<i>vEWrt</i>	<i>vMWrt</i>	<i>pcWrt</i>		
0	1	1	0	0	0		

## Cycle 13: Execute

<i>nRegid</i>	<i>nValC</i>	<i>regA</i>	<i>regB</i>	<i>regE</i>	<i>regM</i>	<i>aluA</i>	<i>aluB</i>
X	X	XX	XX	00	00	00	1
<i>aluF</i>	<i>setCnd</i>	<i>mAddr</i>	<i>mData</i>	<i>mRd</i>	<i>mWrt</i>	<i>newPC</i>	<i>vPWrt</i>
00	0	X	X	X	0	XX	0
<i>irWrt</i>	<i>vAWrt</i>	<i>vBWrt</i>	<i>vEWrt</i>	<i>vMWrt</i>	<i>pcWrt</i>		
0	0	0	1	0	0		

## Cycle 14: Memory

<i>nRegid</i>	<i>nValC</i>	<i>regA</i>	<i>regB</i>	<i>regE</i>	<i>regM</i>	<i>aluA</i>	<i>aluB</i>
X	X	XX	XX	00	00	XX	X
<i>aluF</i>	<i>setCnd</i>	<i>mAddr</i>	<i>mData</i>	<i>mRd</i>	<i>mWrt</i>	<i>newPC</i>	<i>vPWrt</i>
XX	0	0	0	0	1	XX	0
<i>irWrt</i>	<i>vAWrt</i>	<i>vBWrt</i>	<i>vEWrt</i>	<i>vMWrt</i>	<i>pcWrt</i>		
0	0	0	0	0	0		

## Cycle 15: Write Back

<i>nRegid</i>	<i>nValC</i>	<i>regA</i>	<i>regB</i>	<i>regE</i>	<i>regM</i>	<i>aluA</i>	<i>aluB</i>
X	X	XX	XX	10	00	XX	X
<i>aluF</i>	<i>setCnd</i>	<i>mAddr</i>	<i>mData</i>	<i>mRd</i>	<i>mWrt</i>	<i>newPC</i>	<i>vPWrt</i>
XX	0	X	X	X	0	XX	0
<i>irWrt</i>	<i>vAWrt</i>	<i>vBWrt</i>	<i>vEWrt</i>	<i>vMWrt</i>	<i>pcWrt</i>		
0	0	0	0	0	0		

## Cycle 16: PC Update

<i>nRegid</i>	<i>nValC</i>	<i>regA</i>	<i>regB</i>	<i>regE</i>	<i>regM</i>	<i>aluA</i>	<i>aluB</i>
X	X	XX	XX	00	00	XX	X
<i>aluF</i>	<i>setCnd</i>	<i>mAddr</i>	<i>mData</i>	<i>mRd</i>	<i>mWrt</i>	<i>newPC</i>	<i>vPWrt</i>
XX	0	X	X	X	0	00	0
<i>irWrt</i>	<i>vAWrt</i>	<i>vBWrt</i>	<i>vEWrt</i>	<i>vMWrt</i>	<i>pcWrt</i>		
0	0	0	0	0	1		

2. The control signal for `imrmov V, rA (C|0|rA|F|V|)`

Cycle 1: Fetch

<i>nRegid</i>	<i>nValC</i>	<i>regA</i>	<i>regB</i>	<i>regE</i>	<i>regM</i>	<i>aluA</i>	<i>aluB</i>
1	1	XX	XX	00	00	XX	X
<i>aluF</i>	<i>setCnd</i>	<i>mAddr</i>	<i>mData</i>	<i>mRd</i>	<i>mWrt</i>	<i>newPC</i>	<i>vPWrt</i>
XX	0	X	X	X	0	XX	1
<i>irWrt</i>	<i>vAWrt</i>	<i>vBWrt</i>	<i>vEWrt</i>	<i>vMWrt</i>	<i>pcWrt</i>		
1	0	0	0	0	0		

Cycle 2: Execute

<i>nRegid</i>	<i>nValC</i>	<i>regA</i>	<i>regB</i>	<i>regE</i>	<i>regM</i>	<i>aluA</i>	<i>aluB</i>
X	X	XX	XX	00	00	10	0
<i>aluF</i>	<i>setCnd</i>	<i>mAddr</i>	<i>mData</i>	<i>mRd</i>	<i>mWrt</i>	<i>newPC</i>	<i>vPWrt</i>
00	0	X	X	X	0	XX	0
<i>irWrt</i>	<i>vAWrt</i>	<i>vBWrt</i>	<i>vEWrt</i>	<i>vMWrt</i>	<i>pcWrt</i>		
0	0	0	1	0	0		

Cycle 3: Memory

<i>nRegid</i>	<i>nValC</i>	<i>regA</i>	<i>regB</i>	<i>regE</i>	<i>regM</i>	<i>aluA</i>	<i>aluB</i>
X	X	XX	XX	00	00	XX	X
<i>aluF</i>	<i>setCnd</i>	<i>mAddr</i>	<i>mData</i>	<i>mRd</i>	<i>mWrt</i>	<i>newPC</i>	<i>vPWrt</i>
XX	0	0	X	1	0	XX	0
<i>irWrt</i>	<i>vAWrt</i>	<i>vBWrt</i>	<i>vEWrt</i>	<i>vMWrt</i>	<i>pcWrt</i>		
0	0	0	0	0	0		

Cycle 4: Write Back

<i>nRegid</i>	<i>nValC</i>	<i>regA</i>	<i>regB</i>	<i>regE</i>	<i>regM</i>	<i>aluA</i>	<i>aluB</i>
X	X	XX	XX	00	10	XX	X
<i>aluF</i>	<i>setCnd</i>	<i>mAddr</i>	<i>mData</i>	<i>mRd</i>	<i>mWrt</i>	<i>newPC</i>	<i>vPWrt</i>
XX	0	X	X	X	0	XX	0
<i>irWrt</i>	<i>vAWrt</i>	<i>vBWrt</i>	<i>vEWrt</i>	<i>vMWrt</i>	<i>pcWrt</i>		
0	0	0	0	0	0		

Cycle 5: PC Update

<i>nRegid</i>	<i>nValC</i>	<i>regA</i>	<i>regB</i>	<i>regE</i>	<i>regM</i>	<i>aluA</i>	<i>aluB</i>
X	X	XX	XX	00	00	XX	X
<i>aluF</i>	<i>setCnd</i>	<i>mAddr</i>	<i>mData</i>	<i>mRd</i>	<i>mWrt</i>	<i>newPC</i>	<i>vPWrt</i>
XX	0	X	X	X	0	00	0
<i>irWrt</i>	<i>vAWrt</i>	<i>vBWrt</i>	<i>vEWrt</i>	<i>vMWrt</i>	<i>pcWrt</i>		
0	0	0	0	0	1		

## 2 Pipelining

1. Show how the sequence of instructions be executed

```

1) rrmov %rax, %rbx
2) rrmov %rax, %rcx
3) rmmov %rbx, 4(%rdx)
4) mrmov 8(%rcx), %rdx
5) add   %rdx, %rbx
6) irmov 5, %rax
7) add   %rax, %rdx

```

Inst	<i>valA</i>	<i>valB</i>	1	2	3	4	5	6	7	8	9	10	11	12
1			F	D	E	M	W							
2				F	D	E	M	W						
3	<i>M_valE</i>				F	D	E	M	W					
4		<i>M_valE</i>				F	D	E	M	W				
5	<i>e_valE</i>						F	D	E	M	W			
6								F	D	E	M	W		
7	<i>e_valE</i>	<i>W_valE</i>							F	D	E	M	W	

2. Program 1 runs faster than Program 2. This is because the repetition structure in Program 1 was designed so that the condition for `jge` is *true* only in the last iteration. This conforms with the *Assume Branch Not Taken* technique to solve the control hazards.

Program1

```

    irmov $10, %rsi
    irmov $0, %rax
    irmov $1, %rdi
L:   cmp   %rsi, %rax
     jge   E
     add   %rdi, %rax
     jmp   L
E:   hlt

```

Program2

```

    irmovl 10, %esi
    irmovl 0, %eax
    irmovl 1, %edi
L:   addl   %edi, %eax
     cmpl   %esi, %eax
     jl     L
     hlt

```

### 3 Cache Memory

1. The hit rate is 75% when the block size is 16 bytes.
2. The hit rate is 93.70% when the block size is 64 bytes (16 integers) since there were  $\lceil \frac{1000}{16} \rceil = 63$  misses in the cache access.

### 4 Vector Processing

```
#include<stdio.h>
#include<stdlib.h>
#include<immintrin.h>
#include<x86intrin.h>
#include<math.h>

#define ALIGN __attribute__((aligned (32)))

int main() {
    int i, n;
    double ALIGN A[] = {1,2,3,4};
    double ALIGN B[] = {4,4,4,4};
    double ALIGN S[] = {0,0,0,0};
    double sum;

    __m256d a, b, s;

    printf("Enter n: ");
    scanf("%d", &n);

    a = _mm256_load_pd(A);
    b = _mm256_load_pd(B);
    s = _mm256_load_pd(S);

    for(i=0; i<n/4; i++) {
        s = _mm256_add_pd(s, a);
        a = _mm256_add_pd(a, b);
    }

    s = _mm256_add_pd(s, _mm256_permute2f128_pd(s, s, 1));
    s = _mm256_add_pd(s, _mm256_permute_pd(s, 5));

    _mm256_store_pd(S, s);
    sum = S[0];

    for(i=n/4*4+1; i<=n; i++) {
        sum += i;
    }
}
```

```
    printf("Sum = %.0lf\n", sum);  
  
    return 1;  
}
```