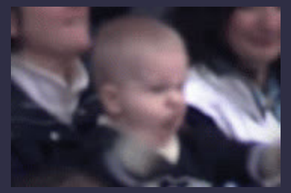
PRÜFUNG

Rechnerarchitektur

Willkommen!

Wir freuen uns, dass du hier bist!



Wichtig

1

Datum: Freitag, 10. Juni 2022

- Zeit: 16:00 Uhr
- Raum: A6 (noch nicht definitiv)
- Dauer: 120 Minuten
- Erlaubte Hilfsmittel:

4 einzelne A4-Seiten oder 2 Doppelseiten.
Handgeschrieben oder auf dem iPad geschrieben und ausgedruckt

- Mitnehmen: Legi
- Anmeldung: KSL (Fristen beachten)
- Voraussetzungen:

Praxis Teil (Serien): 4 von 5 Punkte
Theorie Teil: mindestens 60% Punkte

- Übungen Bonus (Raspberry PI):

+10% (Theorie Teil)

1 MULTIPLE-CHOICE QUESTIONS (5 POINTS)

(a) `rs`, `rt` and `rd` use 5 bits which are enough to address all 32 registers.

○ True ○ False **Solution:** True

(b) The instructions `add`, `sub` and `or` are all of type R.

○ True ○ False **Solution:** True

(c) In a MIPS pipeline architecture, the execution time of a program is exactly equal to the number of instructions for a program times the clock cycle time.

○ True ○ False **Solution:** False

(d) CISC instructions typically have a fixed size.

○ True ○ False **Solution:** False. CISC typically uses variable length instructions.

(e) In a processor with a pipeline architecture, state registers are used to isolate the pipeline stages.

○ True ○ False **Solution:** True.

I-Format:

op	rs	rt	immediate
001000	10011	01010	0000000000000100

Example: `addi $t2, $s3, 4`

J-Format:

op	address
000010	00000000000000000100000001

Example: `j LOOP` (or `j 1028`)

R-Format:

op	rs	rt	rd	shamt	funct
000000	10001	10010	10000	00000	100000

Example: `add $s0, $s1, $s2`

#02
MIPS SINGLE-CYCLE DATAPATH (17 POINTS

#02
MIPS SINGLE-CYCLE DATAPATH (17 POINTS

Consider the single-cycle implementation of a MIPS processor depicted in the image below. Suppose that the processor executes the instruction `ori $r20,$r23,0xc1`. The `$PC` register holds `0x1004'0004`. The state of the register file is given in the following table:

Register file	
Register	Content
`$r20`	`0x0000'0035`
`$r21`	`0x0000'1042`
`$r22`	`0x0000'1040`
`$r23`	`0x0000'1018`

Based on the given implementation and register contents, answer the questions below.
Note: For all the tasks the `0x` prefix is used to indicate hexadecimal values.

(a) (1 point) What value does the `$PC` register hold after executing the instruction?

#02
MIPS SINGLE-CYCLE DATAPATH (17 POINTS

Consider the single-cycle implementation of a MIPS processor depicted in the image below. Suppose that the processor executes the instruction `ori $r20,$r23,0xc1`. The `$PC` register holds `0x1004'0004`. The state of the register file is given in the following table:

Register file	
Register	Content
`$r20`	`0x0000'0035`
`$r21`	`0x0000'1042`
`$r22`	`0x0000'1040`
`$r23`	`0x0000'1018`

Based on the given implementation and register contents, answer the questions below.
Note: For all the tasks the `0x` prefix is used to indicate hexadecimal values.

(a) (1 point) What value does the `$PC` register hold after executing the instruction?

Solution:
PC + 4 = 0x1004'0008

#02
MIPS SINGLE-CYCLE DATAPATH (17 POINTS

(b) (5 points) How is the instruction `ori $r20,$r23,0xc1` encoded in the instruction memory? State the binary representation of this instruction and describe your solution process. The opcode of `ori` is `0xd`. *Hint:* `ori` is an I-type instruction, the syntax of an `ori` instruction is `ori $rt,$rs,imm`. The performed operation is `R[rt] = R[rs] | imm`.

#02
MIPS SINGLE-CYCLE DATAPATH (17 POINTS

(b) (5 points) How is the instruction `ori $r20,$r23,0xc1` encoded in the instruction memory? State the binary representation of this instruction and describe your solution process. The opcode of `ori` is `0xd`. *Hint:* `ori` is an I-type instruction, the syntax of an `ori` instruction is `ori $rt,$rs,imm`. The performed operation is `R[rt] = R[rs] | imm`.

Solution:
(penalty if encoding not in binary)
I-Type format: opcode(6) | rs(5) | rt(5) | imm(16)
(1 point) opcode = 0xd = 0b00'1101
(1 point) rs = 23 = 0x17 = 0b0001'0111
(1 point) rt = 20 = 0x14 = 0b0001'0100
(1 point) imm = 193 = 0xc1 = 0b1100'0001
(1 point) encoding: 0xd | 0x14 | 0x17 | 0xc1 = 0b001101 | 10111 | 10100 | 0000 0000 1100 0001

#02
MIPS SINGLE-CYCLE DATAPATH (17 POINTS

(c) (4 points) What are the values of the control signals `RegDst`, `Jump`, `Branch`, `MemRead`, `MemtoReg`, `MemWrite`, `ALUsrc` and `RegWrite`? Also state "Do not Care" cases, if any, with "X".

RegDst	Jump	Branch	MemRead
MemtoReg	**MemWrite**	**ALUsrc**	**RegWrite**

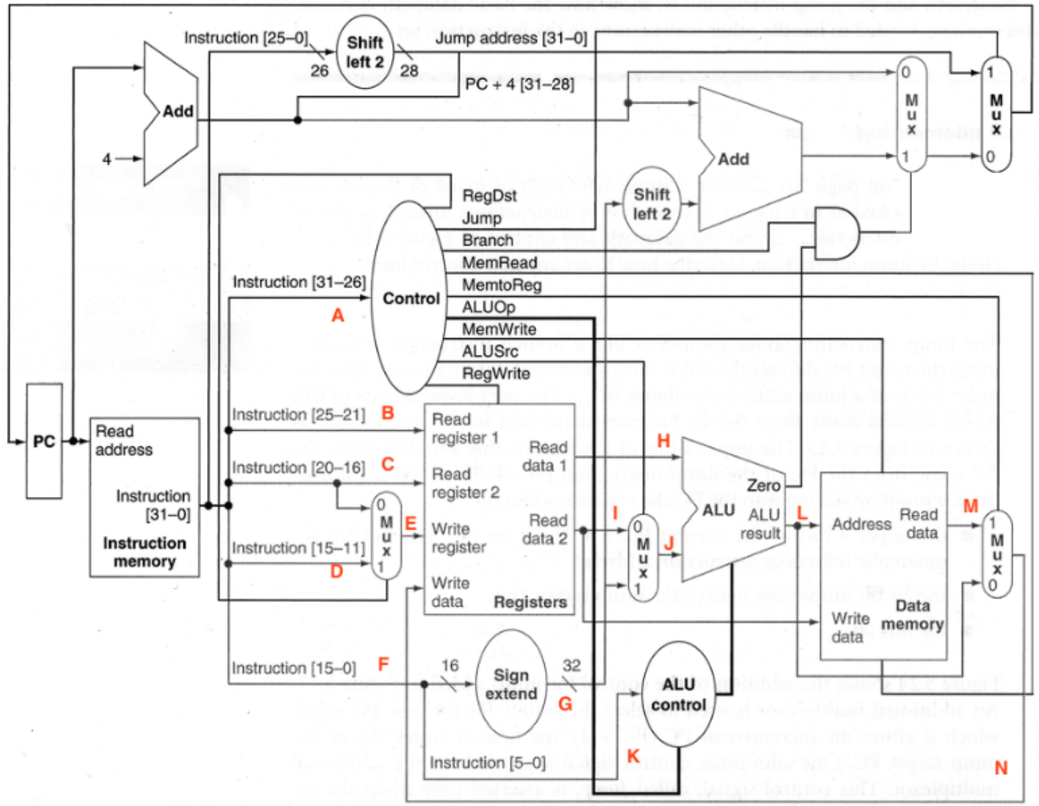
#02
MIPS SINGLE-CYCLE DATAPATH (17 POINTS

(c) (4 points) What are the values of the control signals `RegDst`, `Jump`, `Branch`, `MemRead`, `MemtoReg`, `MemWrite`, `ALUsrc` and `RegWrite`? Also state "Do not Care" cases, if any, with "X".

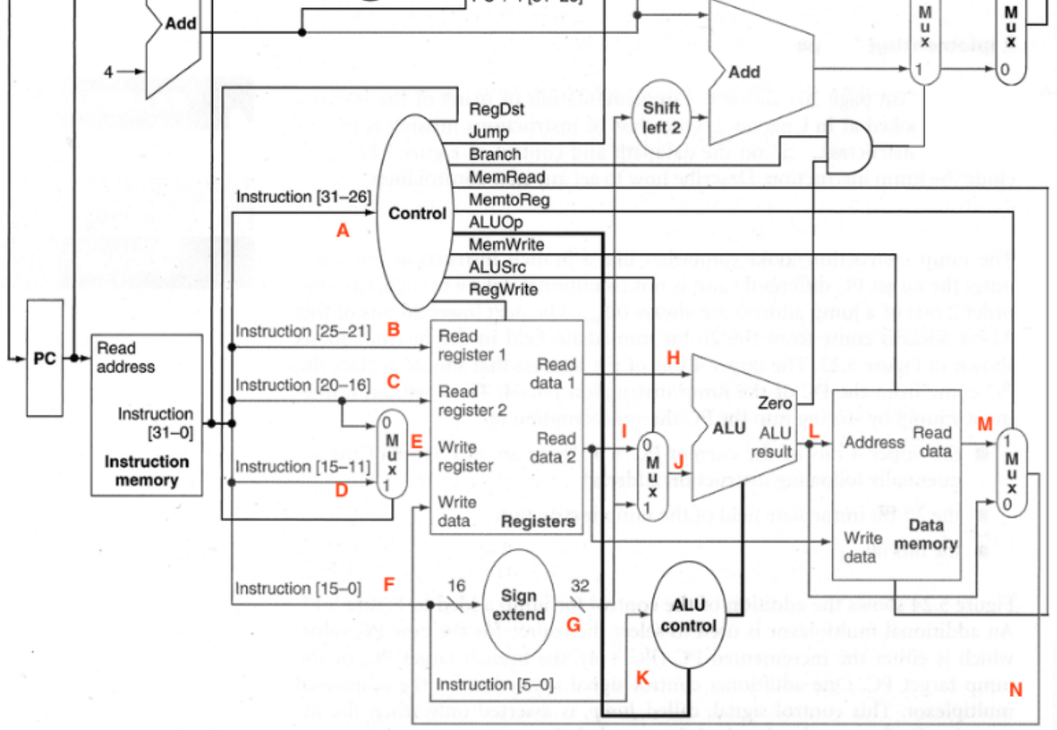
RegDst	**Jump**	**Branch**	**MemRead**
MemtoReg	**MemWrite**	**ALUsrc**	**RegWrite**

Solution: (0.5 points each)
0 0 0 0
0 0 1 1

(d) (7 points) For each letter in the diagram, state the value of the signal on the corresponding wire. You may use binary, hexadecimal or decimal values. All (if any) undefined signals are to be marked with "X".



A	B	C	D
E	**F**	**G**	**H**
I	**J**	**K**	**L**
M	**N**		



A	B	C	D
E	**F**	**G**	**H**
I	**J**	**K**	**L**
M	**N**		

Solution: (0.5 points each)

A 0xd (opcode)	**B** 0x14 (rs)	**C** 0x17 (rt)	**D** 0x0
E C (I-Type, mux=0)	**F** 0xc1	**G** 0xFFFFFFc1 (sign ext.)	**H** 0x0000'1018 ($r23)
I X	**J** G (I-Type, mux=1)	**K** 0x1	**L** H | J (bitwise or)
M X	**N** (mux=0)		

#03
C PROGRAMMING
(12 POINTS.)

(a) (8 points) Write a C function that returns the largest prime number in an array. Choose an appropriate return value in case the array contains no prime numbers. The function declaration should look as follows:

```
int largestPrime(int array[], int size)
```

You can assume you are given the implementation of a function `int isPrime(int x) { ... }` that returns 1 (true) if `x` is prime and 0 (false) otherwise.

Solution:

```
int largestPrime(int array[], int size) {
        int largest = 0; // if no primes, return 0: 1 point
        int i; // 1 point
        for(i = 0; i < size; i++) { // 2 points for correct signature, 1 point for minor mistak
                if (isPrime(array[i]) && (array[i] > largest)) {
                        // logical AND or two if clauses: 2 points
                        // use of isPrime: 1 point
                        largest = array[i];
                }
        }

        return largest; // 1 point
}


// additional points if maximum not already reached: 1 point
int main() {
        int x[6] = {1, 2, 3, 4, 5, 6};
        printf("%d", largestPrime(x, sizeof(x) / sizeof(x[0])));
}
```

(b) (2 points) Why do we need to pass the size of the array as an argument in the above function?

(b) (2 points) Why do we need to pass the size of the array as an argument in the above function?

Solution: The array is referenced by a pointer to the first element. Therefore, the size is unknown.

(c) (2 points) How would you change the declaration of this function such that it operates on a pointer to integers?

(c) (2 points) How would you change the declaration of this function such that it operates on a pointer to integers?

Solution: `int largestPrime(int *array, int size)`

#04
MIPS (10 POINTS)

#04 MIPS (10 POINTS)

THE FOLLOWING MIPS ASSEMBLY PROGRAM IS READING FROM INTEGER ARRAYS A AND B AND WRITING TO THE INTEGER ARRAY A. ASSUME THE BASE ADDRESS OF A AND B IS SAVED IN REGISTER $T0 AND $T1, RESPECTIVELY, AND THAT SIZEOF(INT) = 4.

#04 MIPS (10 POINTS)

```
1   addi $t2, $zero, 4              // i = 4
2   addi $t3, $zero, 2
3   sw $t3, 0($t0)                  // a[0] = _________________
4
5   loop:   lw $t3, 0($t0)          // read a[0]
6
7           beq $t2, 16, end
8
9           addi $t2, $t2, 4        // i++
10
11          add $t4, $t2, $t1       // $t4 = i + base address of b
12          lw $t5, 0($t4)          // $t5 = b[i]
13          add $t3, $t5, $t3       // $t3 = b[i] + a[0]
14
15          add $t5, $t2, $t0
16          sw $t3, 0($t5)          // a[i] = _________________
17
18          j loop
19
20  end:    sw $zero, 0($t0)        // a[0] = _________________
```

THE FOLLOWING MIPS ASSEMBLY PROGRAM IS READING FROM INTEGER ARRAYS A AND B AND WRITING TO THE INTEGER ARRAY A. ASSUME THE BASE ADDRESS OF A AND B IS SAVED IN REGISTER $T0 AND $T1, RESPECTIVELY, AND THAT SIZEOF(INT) = 4.

THE FOLLOWING MIPS ASSEMBLY PROGRAM IS READING FROM INTEGER ARRAYS A AND B AND WRITING TO THE INTEGER ARRAY A. ASSUME THE BASE ADDRESS OF A AND B IS SAVED IN REGISTER $T0 AND $T1, RESPECTIVELY, AND THAT SIZEOF(INT) = 4.

#04 MIPS (10 POINTS)

(a) (3 points) Fill the three blank lines 3, 16, and 20 in the comments above with suitable C pseudo code.

Solution:

THE FOLLOWING MIPS ASSEMBLY PROGRAM IS READING FROM INTEGER ARRAYS A AND B AND WRITING TO THE INTEGER ARRAY A. ASSUME THE BASE ADDRESS OF A AND B IS SAVED IN REGISTER $T0 AND $T1, RESPECTIVELY, AND THAT SIZEOF(INT) = 4.

#04 MIPS (10 POINTS)

(a) (3 points) Fill the three blank lines 3, 16, and 20 in the comments above with suitable C pseudo code.

Solution:

```
1  (3) a[0] = 2;
2  (16) a[i] = b[i] + a[0];
3  (20) a[0] = 0;
```

(1 point each)

THE FOLLOWING MIPS ASSEMBLY PROGRAM IS READING FROM INTEGER ARRAYS A AND B AND WRITING TO THE INTEGER ARRAY A. ASSUME THE BASE ADDRESS OF A AND B IS SAVED IN REGISTER $T0 AND $T1, RESPECTIVELY, AND THAT SIZEOF(INT) = 4.

#04 MIPS (10 POINTS)

(b) (1 point) How many times does the loop iterate?

THE FOLLOWING MIPS ASSEMBLY PROGRAM IS READING FROM INTEGER ARRAYS A AND B AND WRITING TO THE INTEGER ARRAY A. ASSUME THE BASE ADDRESS OF A AND B IS SAVED IN REGISTER $T0 AND $T1, RESPECTIVELY, AND THAT SIZEOF(INT) = 4.

#04 MIPS (10 POINTS)

(b) (1 point) How many times does the loop iterate?

Solution: The register `$t2` (the counter variable) initially holds the value 4. It is incremented by 3 after the branch instruction until it reaches the value 16. Therefore, the loop is exectued four times.

THE FOLLOWING MIPS ASSEMBLY PROGRAM IS READING FROM INTEGER ARRAYS A AND B AND WRITING TO THE INTEGER ARRAY A. ASSUME THE BASE ADDRESS OF A AND B IS SAVED IN REGISTER $T0 AND $T1, RESPECTIVELY, AND THAT SIZEOF(INT) = 4.

#04 MIPS (10 POINTS)

(c) (6 points) Translate the above MIPS program to C code.

#04 MIPS (10 POINTS)

```
1  addi $t2, $zero, 4              // i = 4
2  addi $t3, $zero, 2
3  sw $t3, 0($t0)                  // a[0] = _________________
4
5  loop:   lw $t3, 0($t0)          // read a[0]
6
7          beq $t2, 16, end
8
9          addi $t2, $t2, 4        // i++
10
11         add $t4, $t2, $t1       // $t4 = i + base address of b
12         lw $t5, 0($t4)          // $t5 = b[i]
13         add $t3, $t5, $t3       // $t3 = b[i] + a[0]
14
15         add $t5, $t2, $t0
16         sw $t3, 0($t5)          // a[i] = _________________
17
18         j loop
19
20 end:    sw $zero, 0($t0)        // a[0] = _________________
```

(c) (6 points) Translate the above MIPS program to C code.

Solution:

```
1  int a[10];
2  int b[10] = { ... }
3
4  a[0] = 2;                          // 1 point
5  int j;                             // optional 0.5 points if maximum not reached
6  for (j = 1; j <= 4; j++) {         // 2 points
7      a[j] = b[j] + a[0];            // 2 points
8  }
9  a[0] = 0;                          // 1 point
```