程序代写代做 CS编程辅导

MIPS

WeChat: cstutorcs

Assignment Project Exam Help

Email: tutorcs@163.com

QQ: 749389476

https://tutorcs.com

Today:

程序代写代做CS编程辅导

Quick overview of the MIPS instruction set.

- We're going to be compiling to MIPS assembly language.
- So you need to know how to program at the MIPS level.
- Helps to have a bit of architecture background to understand *why* MIPS assembly is the way it is.

There's an online manual that describes things in gory detail.

WeChat: cstutorcs
Assignment Project Exam Help
Email: tutorcs@163.com
QQ: 749389476
https://tutorcs.com

Assembly vs Machine Code

程序代写代做CS编程辅导

- We write assembly language instructions
  - e.g., "**addi r2, 42**"
- The machine interprets machine code *bits*
  - e.g., "**1010110011001111**…"
  - Your next assignment is to build an interpreter for a subset of the MIPS machine code.
- The assembler takes care of compiling assembly language to bits for us.
  - It also provides a few conveniences as we'll see.

WeChat: cstutorcs
Assignment Project Exam Help
Email: tutorcs@163.com
QQ: 749389476
https://tutorcs.com

Some MIPS Assembly

程序代写代做 CS编程辅导

```
int sum(int n)
  int s = 0;
  for (; n != 0; n--)
    s += n;
}
```

```
sum:   ori  $2,$0,$0
       b    test
loop:  add  $2,$2,$4
       subi $4,$4,1
test:  bne  $4,$0,loop
       j    $31
```

WeChat: cstutorcs

Assignment Project Exam Help

Email: tutorcs@163.com

QQ: 749389476

https://tutorcs.com

```
int main() {
  return sum(42);
}
```

```
main:  ori  $4,$0,42
       move $17,$31
       jal  sum
       jr   $17
```

An X86 Example (-O0):

程序代写代做CS编程辅导

```
_sum:
  pushq%rbp
  movq %rsp, %rbp
  movl %edi, -4(%rbp)
  movl $0, -12(%rbp)
  jmp  LBB1_2
LBB1_1:
  movl -12(%rbp), %eax
  movl -4(%rbp), %ecx
  addl %ecx, %eax
  movl %eax, -12(%rbp)
  movl -4(%rbp), %eax
  subl $1, %eax
  movl %eax, -4(%rbp)
LBB1_2:
  movl -4(%rbp), %eax
  cmpl $0, %eax
  jne  LBB1_1
  movl -8(%rbp), %eax
  popq %rbp
  ret
```

WeChat: cstutorcs

Assignment Project Exam Help

Email: tutorcs@163.com

QQ: 749389476

https://tutorcs.com

```
_main:
    pushq  %rbp
    movq   %rsp, %rbp
    subq   $16, %rsp
    movl   $42, %eax
    movl   %eax, %edi
    callq  _sum
    movl   %eax, %ecx
    movl   %ecx, -8(%rbp)
    movl   -8(%rbp), %ecx
    movl   %ecx, -4(%rbp)
    movl   -4(%rbp), %eax
    addq   $16, %rsp
    popq   %rbp
    ret
```

An X86 Example (-O3):

程序代写代做CS编程辅导

```
_sum:
  pushq   %rbp
  movq %rsp, %rbp
  popq %rbp
  ret

_main:
  pushq   %rbp
  movq %rsp, %rbp
  popq %rbp
  ret
```

WeChat: cstutorcs

Assignment Project Exam Help

Email: tutorcs@163.com

QQ: 749389476

https://tutorcs.com

MIPS

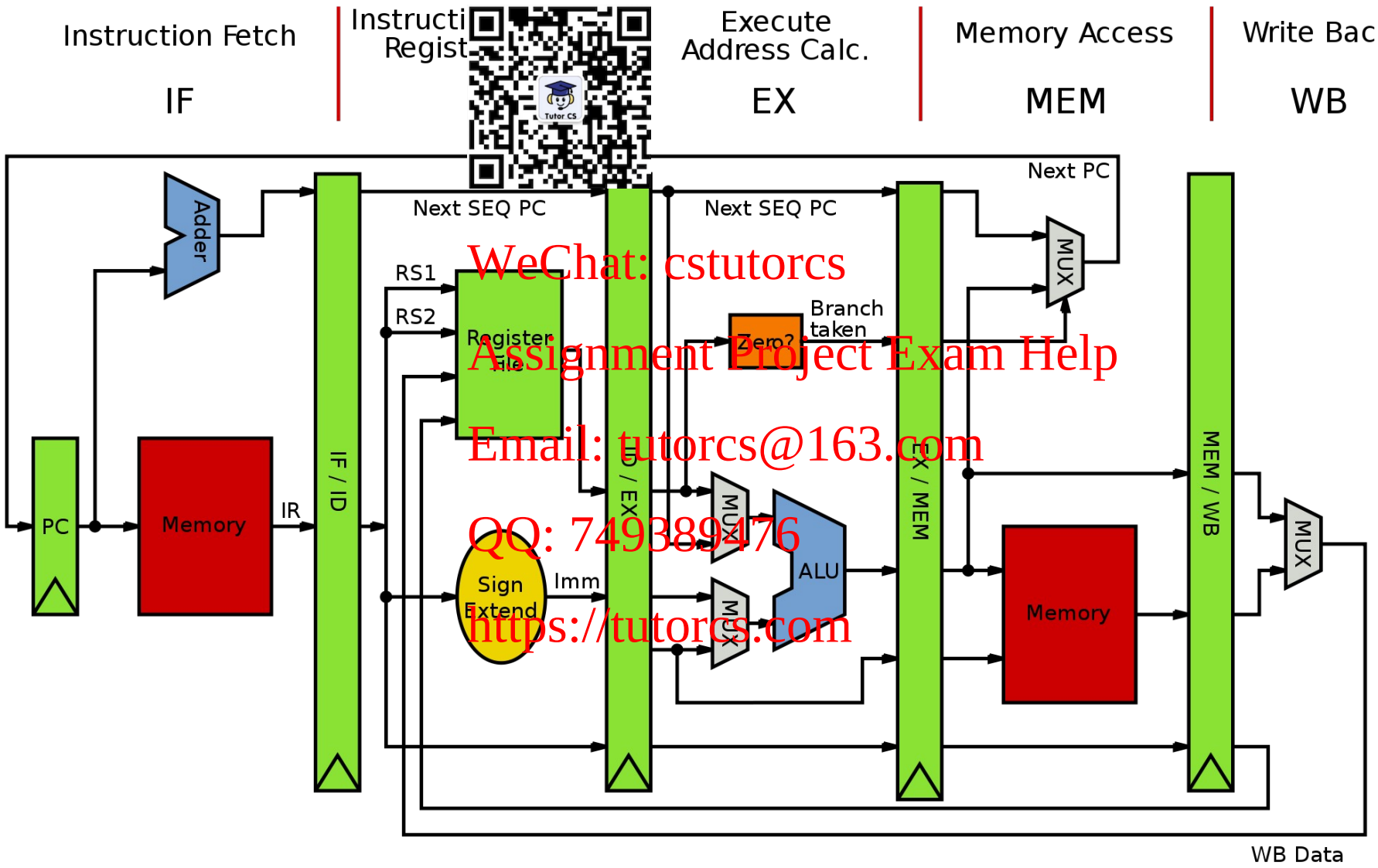
程序代写代做CS编程辅导

- Reduced Instruction Set Computer (RISC)
  - Load/store architecture
  - All operands are either registers or constants
  - All instructions same size (4 bytes) and aligned on 4-byte boundary
  - Simple, orthogonal instructions
    - e.g., no **subi**, (**addi** and negate value)
  - All registers (except $0) can be used in all instructions.
    - Reading $0 always returns the value 0
- Easy to make fast: pipeline, superscalar

WeChat: cstutorcs
Assignment Project Exam Help
Email: tutorcs@163.com
QQ: 749389476
https://tutorcs.com

MIPS Datapath

程序代写代做CS编程辅导



WeChat: cstutorcs
Assignment Project Exam Help
Email: tutorcs@163.com
QQ: 749389476
https://tutorcs.com

x86

程序代写代做CS编程辅导

- Complex Instruction Set Computer (CISC)
  - Instructions operate on memory values
    - e.g., add [eax],ebx
  - Complex, multi-cycle instructions
    - e.g., string copy, call
  - Many ways to do the same thing
    - e.g., add eax,1 inc eax, sub eax,-1
  - Instructions are variable-length (1-10 bytes)
  - Registers are not orthogonal
- Hard to make fast…(but they do anyway)

WeChat: cstutorcs
Assignment Project Exam Help
Email: tutorcs@163.com
QQ: 749389476
https://tutorcs.com

Tradeoffs

程序代写代做 CS编程辅导

- x86 (as opposed to RISC/MIPS):
  - Lots of existing code.
  - Harder to decode (i.e., parse).
  - Harder to assemble/compile to.
  - Code can be more compact (3 bytes on avg.)
  - I-cache is more effective…
  - Easier to add new instructions.
- Today's implementations have the best of both:
  - Intel & AMD chips take in x86 instructions and remap them to "micro-ops", caching the results.
  - Core execution engine more like MIPS.

WeChat: cstutorcs
Assignment Project Exam Help
Email: tutorcs@163.com
QQ: 749389476
https://tutorcs.com

MIPS instructions:

程序代写代做CS编程辅导

- Arithmetic & logical instructions:
  - **add**, **sub**, **and**, **sll**, **srl**, **sra**, …
  - Register and immediate forms:
  - **add** $*rd*, $*rs*, $*rt*
  - **addi** $*rd*, $*rs*, *<16-bit-immed>*
  - Any registers (except $0 returns 0)
  - Also a distinction between overflow and no-overflow (we'll ignore for now.)

WeChat: cstutorcs
Assignment Project Exam Help
Email: tutorcs@163.com
QQ: 749389476
https://tutorcs.com

Encodings:

程序代写代做CS编程辅导

add $*rd*, $*rs*, $*rt*

Op1:6	rs:5	rt:5	rd:5	0:5	Op2:6

WeChat: cstutorcs

Assignment Project Exam Help

Email: tutorcs@163.com

QQ: 749389476

https://tutorcs.com

addi $*rt*, $*rs*, <*imm*>

Op1:6	rs:5	rt:5	imm:16

Movement:

程序代写代做CS编程辅导

- Assembler provides pseudo-instructions:

  move $*rd*, $*rs* → **or** $*rd*, $*rs*, $0

  li $*rd*, *<32-bit imm>* →
  lui $*rd*, *<hi-16-bits>*
  ori $*rd*, $*rd*, *<lo-16-bits>*

WeCat: cstutorcs
Assignment Project Exam Help
Email: tutorcs@163.com
QQ: 749389476
https://tutorcs.com

MIPS instructions:

程序代写代做CS编程辅导

- Multiply and D
  - Use two spe sters **mflo**, **mfhi**
  - i.e., **mul** $3, $5 produces a 64-bit value which is placed in **mfhi** and **mflo**.
  - Instructions to move values from **mflo**/**hi** to the general purpose registers $r and back.
  - Assembler provides pseudo-instructions:
  - **mult** $2, $3, $5 expands into:

```
mul  $3,$5
mflo $2
```

WeChat: cstutorcs
Assignment Project Exam Help
Email: tutorcs@163.com
QQ: 749389476
https://tutorcs.com

MIPS instructions:

程序代写代做CS编程辅导

- Load/store
  - **`lw`** $*rd*, <*imm*>($*rs*) ; *rd* := Mem[*rs*+*imm*]
  - **`sw`** $*rs*, <*imm*>($*rt*) ; Mem[*rt*+*imm*] := rs
- Traps (fails) if *rs*+*imm* is not word-aligned.
- Other instructions to load bytes and half-words.

WeChat: cstutorcs
Assignment Project Exam Help
Email: tutorcs@163.com
QQ: 749389476
https://tutorcs.com

Conditional Branching:

程序代写代做CS编程辅导

- **beq** $*rs*,$*rt*,<*imm16*>
  if $*rs* == $*rt* then pc := pc + *imm16*
- **bne** $rs,$rt, <*imm16*>
- **b** <*imm16*> == **beq** $0,$0, <*imm16*>
- **bgez** $*rs*, <*imm16*>
  if $*rs* >= 0 then pc := pc + *imm16*
- Also **bgtz**, **blez**, **bltz**

WeChat: cstutorcs
Assignment Project Exam Help
Email: tutorcs@163.com
QQ: 749389476
https://tutorcs.com

In Practice:

程序代写代做 CS编程辅导

Assembler lets you use symbolic labels instead of having to calculate the offsets.

WeChat: cstutorcs

Just as in BASIC, you put a label on an instruction and then can branch to it:

Assignment Project Exam Help

Email: tutorcs@163.com

QQ: 749389476

LOOP: …

bne $3, $2, **LOOP**

https://tutorcs.com

Assembler figures out actual offsets.

Tests:

程序代写代做 CS编程辅导

- **`slt`** $*rd*, $*rs*, $*rt*; rd := (rs < rt)
- **`slt`** $*rd*, $*rs*, *<imm16>*
- Additionally: **`sltu`**, **`slti`**, **`sltiu`**
- Assembler provides pseudo-instructions for **`seq`**, **`sge`**, **`sgeu`**, **`sgt`**, **`sne`**, …

WeChat: cstutorcs
Assignment Project Exam Help
Email: tutorcs@163.com
QQ: 749389476
https://tutorcs.com

Unconditional Jumps:

程序代写代做CS编程辅导

- **`j`** <*imm26*> = (imm26 << 2)
- **`jr`** $*rs* = $rs
- **`jal`** <*imm26*> ; $31 := pc+4 ;
  pc := (imm26 << 2)
- Also, **`jalr`** and a few others.
- Again, in practice, we use labels:

```
fact:  …
main:  …
       jal fact
```

WeChat: cstutorcs
Assignment Project Exam Help
Email: tutorcs@163.com
QQ: 749389476
https://tutorcs.com

Other Kinds of Instructions:

程序代写代做CS编程辅导

- Floating-poi arate registers $*fi*)
- Traps
- OS-trickery

WeChat: cstutorcs

Assignment Project Exam Help

Email: tutorcs@163.com

QQ: 749389476

https://tutorcs.com

Our Example:

程序代写代做CS编程辅导

```
int  sum(int n)
  int s = 0;
  for (; n != 0; n--)
    s += n;
}
```

```
sum:   ori  $2,$0,$0
       b    test
loop:  add  $2,$2,$4
       subi $4,$4,1
test:  bne  $4,$0,loop
       jr   $31
```

WeChat: cstutorcs

Assignment Project Exam Help

Email: tutorcs@163.com

QQ: 749389476

https://tutorcs.com

```
int main() {
  return sum(42);
}
```

```
main:  ori  $4,$0,42
       move $17,$31
       jal  sum
       jr   $17
```

Better:

程序代写代做CS编程辅导

```
int  sum(int n)
  int s = 0;
  for (; n != 0; n--)
    s += n;
}
```

```
sum:   ori  $2,$0,$0
       b    test
loop:  add  $2,$2,$4
       subi $4,$4,1
test:  bne  $4,$0,loop
       jr   $31
```

WeChat: cstutorcs
Assignment Project Exam Help
Email: tutorcs@163.com
QQ: 749389476
https://tutorcs.com

```
int main() {
  return sum(42);
}
```

```
main:  ori  $4,$0,42
       j sum
```

One Final Point

程序代写代做CS编程辅导

We're going to program to the MIPS *virtual* machine which is provided by the assembler.

- lets us use macro instructions, labels, etc.
- (but we must leave a scratch register for the assembler to do its work.)
- lets us ignore *delay slots*.
- (but then we pay the price of not scheduling those slots.)

WeChat: cstutorcs
Assignment Project Exam Help
Email: tutorcs@163.com
QQ: 749389476
https://tutorcs.com