PyMTL/Pydgin Tutorial Schedule

8:30am – 8:50am	Virtual Machine Installation and Setup
8:50am – 9:00am	*Presentation:* PyMTL/Pydgin Tutorial Overview
9:00am – 9:10am	*Presentation:* Introduction to Pydgin
9:10am – 10:00am	*Hands-On:* Adding a GCD Instruction using Pydgin
10:00am – 10:10am	*Presentation:* Introduction to PyMTL
10:10am – 11:00am	*Hands-On:* PyMTL Basics with Max/RegIncr
11:00am – 11:30am	Coffee Break
11:30am – 11:40am	*Presentation:* Multi-Level Modeling with PyMTL
11:40am – 12:30pm	*Hands-On:* FL, CL, RTL Modeling of a GCD Unit

★ Task 1.0: Write Euclid's greatest common divisor (GCD) algorithm in Python interpreter ★

```
% ipython
>>> def gcd( a, b ):
...   while b:
...     a, b = b, a%b
...   return a
...
>>> gcd( 1, 5 )
1
>>> gcd( 9, 3 )
3
>>> gcd( 9, 6 )
3
>>> exit()
```

PARC Overview

- MIPS-like 32-bit RISC ISA
- 32 general-purpose registers
- ”Simple” instruction encodings
- Example instruction: add unsigned

```
addu rd, rs, rt                R[rd] = R[rs] + R[rt]

 31     26 25    21 20    16 15    11 10     6 5        0
+--------+-------+-------+-------+-------+--------+
|   op   |  rs   |  rt   |  rd   |       |  cmd   |
| 000000 | src0  | src1  | dst   | 00000 | 100001 |
+--------+-------+-------+-------+-------+--------+
```

Pydgin Framework Overview

- `pydgin/`
  - Common, ISA-independent components
  - Fetch-decode-execute loop, bitwise operations, register file, memory, syscall implementations, JIT annotations
  - A user shouldn't need to modify these

- `parc/`, `arm/`, `<your favorite isa>/`
  - Architecture implementation
  - `machine.py`: architectural state
  - `instruction.py`: static instruction fields
  - `isa.py`: instruction encodings and semantics
  - `parc-sim.py`: executable

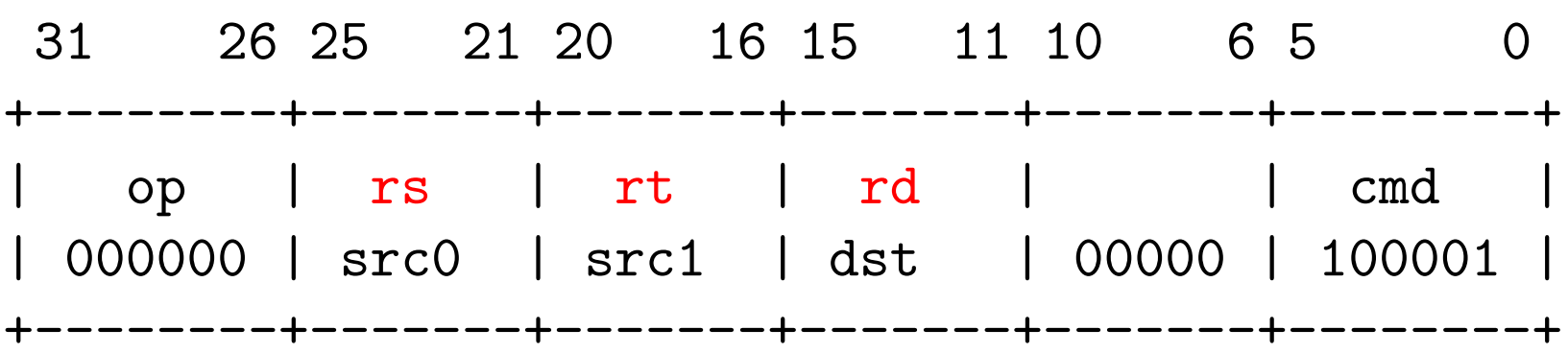
parc/machine.py

```
# the architectural state (simplified)
class State():
  def __init__( self, ... ):
    self.pc  = ...
    self.rf  = ...
    self.mem = ...

    # statistics
    self.num_insts      = ...
    self.stat_num_insts = ...
```

parc/instruction.py

```
addu rd, rs, rt                R[rd] = R[rs] + R[rt]
```




```
class Instruction():
  # ...
  @property
  def rd( self ):
    return (self.bits >> 11) & 0x1F
```

parc/isa.py

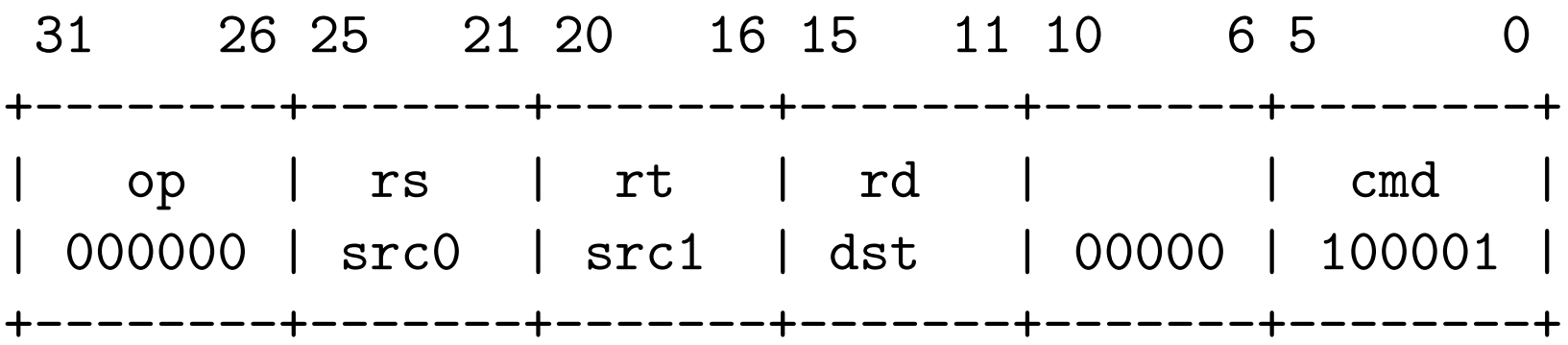
```
 addu rd, rs, rt                R[rd] = R[rs] + R[rt]

 31     26 25    21 20    16 15    11 10     6 5        0
+--------+-------+-------+-------+-------+--------+
|   op   |  rs   |  rt   |  rd   |       |  cmd   |
| 000000 | src0  | src1  | dst   | 00000 | 100001 |
+--------+-------+-------+-------+-------+--------+
```

```
 # instruction encodings
 encodings = [
   # ...
   ['addu', '000000_xxxxx_xxxxx_xxxxx_00000_100001'],
   # ...
 ]
```

parc/isa.py (continued)

```
addu rd, rs, rt                R[rd] = R[rs] + R[rt]
```




```
# addu semantics
# s    = state
# inst = instruction bits
def execute_addu( s, inst ):
  s.rf[ inst.rd ] = trim_32( s.rf[ inst.rs ] + s.rf[ inst.rt ] )
  s.pc += 4
```

Hands-On: Add and evaluate GCD inst in Pydgin

- Given: Extend the cross-compiler
- Given: Add assembly test
- Task 1.1: Build and run assembly tests using cross-compiler
- Task 1.2: Investigate test failures
- Task 1.3: Add encoding for `gcd` instruction
- Task 1.4: Implement semantics
- Task 1.5: Use the debug flags
- Task 1.6: Count the GCD cycles
- Task 1.7: Add inline assembly to application
- Task 1.8: Translate and evaluate

Hands-On: Add and evaluate GCD inst in Pydgin

- **Given: Extend the cross-compiler**
- **Given: Add assembly test**
- Task 1.1: Build and run assembly tests using cross-compiler
- Task 1.2: Investigate test failures
- Task 1.3: Add encoding for `gcd` instruction
- Task 1.4: Implement semantics
- Task 1.5: Use the debug flags
- Task 1.6: Count the GCD cycles
- Task 1.7: Add inline assembly to application
- Task 1.8: Translate and evaluate

Hands-On: Add and evaluate GCD inst in Pydgin

- Given: Extend the cross-compiler
- Given: Add assembly test
- **Task 1.1: Build and run assembly tests using cross-compiler**
- **Task 1.2: Investigate test failures**
- Task 1.3: Add encoding for `gcd` instruction
- Task 1.4: Implement semantics
- Task 1.5: Use the debug flags
- Task 1.6: Count the GCD cycles
- Task 1.7: Add inline assembly to application
- Task 1.8: Translate and evaluate

★ Task 1.1: Build and run assembly tests using cross-compiler ★

```
% cd ~/pydgin-tut/parc/asm_tests/build
% make
```

All tests should pass except for `gcd`:

```
% make check

parc-gcd.out:Exception in execution (pc: 0x00808008), aborting!
parc-gcd.out:Exception message: Invalid instruction 0x9c411811!
```

★ Task 1.2: Investigate test failures ★

```
parc-gcd.out:Exception in execution (pc: 0x00808008), aborting!
parc-gcd.out:Exception message: Invalid instruction 0x9c411811!

% cd ~/pydgin-tut/parc/asm_tests/build
% maven-objdump -dC parc-gcd > gcd.dump
% less gcd.dump

  808000:       24010018        li      at,24
  808004:       24020064        li      v0,100
  808008:       9c411811        gcd     v1,v0,at
  80800c:       24040004        li      a0,4
  808010:       241d000e        li      sp,14
  808014:       14640032        bne     v1,a0,8080e0 <_fail>

% less ~/pydgin-tut/parc/asm_tests/parc/parc-gcd.S
```

Hands-On: Add and evaluate GCD inst in Pydgin

- Given: Extend the cross-compiler
- Given: Add assembly test
- Task 1.1: Build and run assembly tests using cross-compiler
- Task 1.2: Investigate test failures
- **Task 1.3: Add encoding for `gcd` instruction**
- **Task 1.4: Implement semantics**
- **Task 1.5: Use the debug flags**
- Task 1.6: Count the GCD cycles
- Task 1.7: Add inline assembly to application
- Task 1.8: Translate and evaluate

★ Task 1.3: Add encoding for `gcd` instruction ★

```
 gcd rd, rs, rt          R[rd] = gcd( R[rs], R[rt] )

 31    26 25   21 20   16 15   11 10    6 5       0
+--------+-------+-------+-------+-------+---------+
|   op   |  rs   |  rt   |  rd   |       |   cmd   |
| 100111 | src0  | src1  | dst   | xxxxx | 010001  |
+--------+-------+-------+-------+-------+---------+
```

```
% cd ~/pydgin-tut/parc
% gedit isa.py
```

```
243  # TASK: pick the correct encoding
244  ['gcd', '111111_11111_11111_11111_11111_111111'],
```

★ Task 1.4: Implement semantics ★

```
% cd ~/pydgin-tut/parc
% gedit isa.py
```

```
342  def execute_addu( s, inst ):
343    s.rf[ inst.rd ] = trim_32( s.rf[ inst.rs ] + s.rf[ inst.rt ] )
344    s.pc += 4
```

```
329  def execute_gcd( s, inst ):
330    # TASK: implement gcd here
331    s.pc += 4
```

```
     def gcd( a, b ):
       while b:
         a, b = b, a%b
       return a
```

WARNING: DO NOT use closures!

```
% cd ~/pydgin-tut/parc/asm_tests/build
% parc-sim.py --test parc-gcd
```

★ Task 1.5: Use the debug flags ★

You can turn on debug flags using a comma-separated list.

- `insts`: instruction dump
- `rf`: registers reads and written to
- `mem`: memory addresses and values reads and written to
- `regdump`: register dump
- `syscalls`: system call information

```
% cd ~/pydgin-tut/parc/asm_tests/build
% parc-sim.py --test --debug insts          parc-gcd | less
% parc-sim.py --test --debug insts,rf,mem   parc-gcd | less
```

```
808000 24010018 addiu    0        :: RD.RF[0 ] = 00000000 :: WR.RF[1 ] = 00000018
808004 24020064 addiu    1        :: RD.RF[0 ] = 00000000 :: WR.RF[2 ] = 00000064
808008 9c411811 gcd      2        :: RD.RF[2 ] = 00000064 :: RD.RF[1 ] = 00000018 :: WR.RF[3 ] = 00000004
80800c 24040004 addiu    3        :: RD.RF[0 ] = 00000000 :: WR.RF[4 ] = 00000004
808010 241d000e addiu    4        :: RD.RF[0 ] = 00000000 :: WR.RF[29] = 0000000e
808014 14640032 bne      5        :: RD.RF[3 ] = 00000004 :: RD.RF[4 ] = 00000004
```

```
% parc-sim.py --test --debug insts,regdump parc-gcd | less
```

Hands-On: Add and evaluate GCD inst in Pydgin

- Given: Extend the cross-compiler
- Given: Add assembly test
- Task 1.1: Build and run assembly tests using cross-compiler
- Task 1.2: Investigate test failures
- Task 1.3: Add encoding for `gcd` instruction
- Task 1.4: Implement semantics
- Task 1.5: Use the debug flags
- **Task 1.6: Count the GCD cycles**
- **Task 1.7: Add inline assembly to application**
- **Task 1.8: Translate and evaluate**

★ Task 1.6: Count the GCD cycles ★

How many times do we loop when executing the `gcd` instruction?

- architectural state (add a new counter called `gcd_ncycles` in `~/pydgin-tut/parc/machine.py:34`)
- GCD semantics (`~/pydgin-tut/parc/isa.py`)
  WARNING: DO NOT use closures!
- uncomment to print the counter (`~/pydgin-tut/parc/parc-sim.py:71`)

```
% cd ~/pydgin-tut/parc/asm_tests/build
% parc-sim.py --test parc-gcd
  [ passed ] parc-gcd
DONE! Status = 0
Instructions Executed = 15
Instructions Executed in Stat Region = 0
Number of cycles in GCD = 4
Total number of cycles (assuming non-GCD CPI=1) = 19
```

★ Task 1.7: Add inline assembly to application ★

```
% cd ~/pydgin-tut/bmarks/ubmark
% gedit ubmark-gcd.cc
```

```
66  int gcd_hw( int a, int b ) {
67    int result;
68    __asm__( "gcd %0, %1, %2" : "=r" (result)
69                              : "r" (a), "r" (b) );
70    return result;
71  }
```

```
76  void gcd_scalar_accel( ... ) {
77    for ( int i = 0; i < size; i++ )
78      dest[i] = gcd_hw( src0[i], src1[i] );
79  }
```

```
% cd ~/pydgin-tut/bmarks/build
% make ubmark-gcd
% parc-sim.py ubmark-gcd --impl all --verify
```

★ Task 1.8: Translate and evaluate ★

```
% cd ~/pydgin-tut/scripts
% ./build.py pydgin-parc-nojit-debug
```

Evaluate using interpretive and translated Pydgin

```
% cd ~/pydgin-tut/bmarks/build
% parc-sim.py                ubmark-gcd --impl scalar
% parc-sim.py                ubmark-gcd --impl scalar-accel
% pydgin-parc-nojit-debug ubmark-gcd --impl scalar
% pydgin-parc-nojit-debug ubmark-gcd --impl scalar-accel
```

If you've implemented everything correctly, you should get 163,513 and 108,414 cycles for `scalar` and `scalar-accel` respectively.