## PyMTL/Pydgin Tutorial Schedule

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>8:30am – 8:50am</td>
<td>Virtual Machine Installation and Setup</td>
</tr>
<tr>
<td>8:50am – 9:00am</td>
<td><strong>Presentation:</strong> PyMTL/Pydgin Tutorial Overview</td>
</tr>
<tr>
<td>9:00am – 9:10am</td>
<td><strong>Presentation:</strong> Introduction to Pydgin</td>
</tr>
<tr>
<td>9:10am – 10:00am</td>
<td><strong>Hands-On:</strong> Adding a GCD Instruction using Pydgin</td>
</tr>
<tr>
<td>10:00am – 10:10am</td>
<td><strong>Presentation:</strong> Introduction to PyMTL</td>
</tr>
<tr>
<td>10:10am – 11:00am</td>
<td><strong>Hands-On:</strong> PyMTL Basics with Max/RegIncr</td>
</tr>
<tr>
<td>11:00am – 11:30am</td>
<td>Coffee Break</td>
</tr>
<tr>
<td>11:30am – 11:40am</td>
<td><strong>Presentation:</strong> Multi-Level Modeling with PyMTL</td>
</tr>
<tr>
<td>11:40am – 12:30pm</td>
<td><strong>Hands-On:</strong> FL, CL, RTL Modeling of a GCD Unit</td>
</tr>
</tbody>
</table>
The Pydgin Framework
The Pydgin ADL: ARMv5 Architectural State

```python
class State( object ):
    def __init__( self, memory, reset_addr=0x400 ):
        self.pc = reset_addr
        self.rf = ArmRegisterFile( self, num_regs=16 )
        self.mem = memory
        self.rf[ 15 ] = reset_addr

        # current program status register (CPSR)
        self.N = 0b0  # Negative condition
        self.Z = 0b0  # Zero condition
        self.C = 0b0  # Carry condition
        self.V = 0b0  # Overflow condition

    def fetch_pc( self ):
        return self.pc
```
The Pydgin ADL: ARMv5 Encodings

encodings = [

['nop', '00000000000000000000000000000000'],
['mul', 'xxxx0000000xxxxxxxxxxxxx1001xxx'],
['umull', 'xxxx0000100xxxxxxxxxxxxx1001xxx'],
['adc', 'xxxx00x0101xxxxxxxxxxxxxxxxxxxxx'],
['add', 'xxxx00x0100xxxxxxxxxxxxxxxxxxxxx'],
['and', 'xxxx0000000xxxxxxxxxxxxxxxxxxxxx'],
['b', 'xxxx1010xxxxxxxxxxxxxxxxxxxxx'],
['bl', 'xxxx1011xxxxxxxxxxxxxxxxxxxxx'],
['bic', 'xxxx00x1110xxxxxxxxxxxxxxxxxxxxx'],
['bkpt', '111000010010xxxxxxxxxxxx0111xxxx'],

# ...

['teq', 'xxxx00x10011xxxxxxxxxxxxxxxxxxxxx'],
['tst', 'xxxx00x10001xxxxxxxxxxxxxxxxxxxxx']
]

Pydgin ADL
State
Encoding Semantics

Pydgin Interpreter
Loop

Pydgin JIT
Annotations

Python
ISS
Script

RPython
Translation Toolchain

Pydgin
ISS
Executable

Pydgin
DBT-ISS
Executable

ISCA 2015
PyMTL/Pydgin Tutorial: Python Frameworks for Highly Productive Computer Architecture Research
The Pydgin ADL: ARMv5 Instruction Semantics

```python
def execute_add( s, inst):
    if condition_passed( s, inst.cond );
        a, = s.rf[ inst.rn ]
        b, _ = shifter_operand( s, inst )
        result = a + b
        s.rf[ inst.rd ] = trim_32(result)

    if inst.S:
        # ...
        s.N = (result >> 31)&1
        s.Z = trim_32(result) == 0
        s.C = carry_from(result)
        s.V = overflow_from(a, b, result)

    if inst.rd == 15:
        return
    s.rf[PC] = s.fetch_pc() + 4
```
An RPython Instruction Set Simulator

```python
def instruction_set_interpreter( memory ):
    state = State( memory )

    while True:
        pc = state.fetch_pc()
        inst = memory[ pc ]  # fetch
        execute = decode( inst )  # decode
        execute( state, inst )  # execute
```

> python iss.py arm_binary
The RPython Translation Toolchain

- Pydgin ADL
  - State
  - Encoding
  - Semantics

- Pydgin Interpreter Loop
- Pydgin JIT Annotations

- Python ISS Script
- RPython Translation Toolchain

- RPython Source
  - Type Inference
  - Optimization
  - Code Generation
  - Compilation

- Compiled Interpreter

> ./pydgin-nojit arm_binary
An RPython ISS with JIT Annotations

```
jd = JitDriver( greens = ['pc'],
               reds = ['state'] )

def instruction_set_interpreter( memory ):
    state = State( memory )

    while True:
        jd.jit_merge_point( s.fetch_pc(), state )

        pc = state.fetch_pc()
        inst = memory[ pc ] # fetch
        execute = decode( inst ) # decode
        execute( state, inst ) # execute

        if state.fetch_pc() < pc:
            jd.can_enter_jit( s.fetch_pc(), state )
```
The RPython Translation Toolchain JIT Generator

Pydgin ADL
State Encoding Semantics

Pydgin Interpreter Loop

Pydgin ISS Script

RPython Translation Toolchain

RPython Source

Type Inference

Optimization

JIT Generator

Code Generation

Compilation

Compiled Interpreter with JIT

> ./pydgin-jit arm_binary
Creating a competitive JIT requires additional RPython JIT hints:

+ Minimal JIT Annotations
+ Elidable Instruction Fetch
+ Elidable Decode
+ Constant Promotion of PC and Memory
+ Word-Based Target Memory
+ Loop Unrolling in Instruction Semantics
+ Virtualizable PC and Statistics

See our paper in ISPASS2015 for detailed information!
Pydgin ISS Performance

![Graph showing Pydgin ISS Performance]