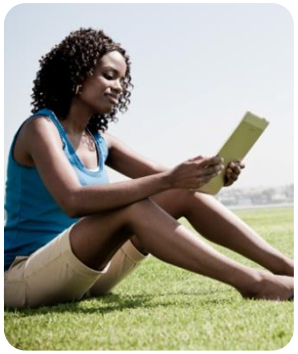
Open *Source* | Open *Possibilities*



# VLIW and the MC Layer

Presented by: Mario Guerra
Qualcomm Innovation Center, Inc

QuIC
QUALCOMM INNOVATION CENTER, INC.

# Introduction

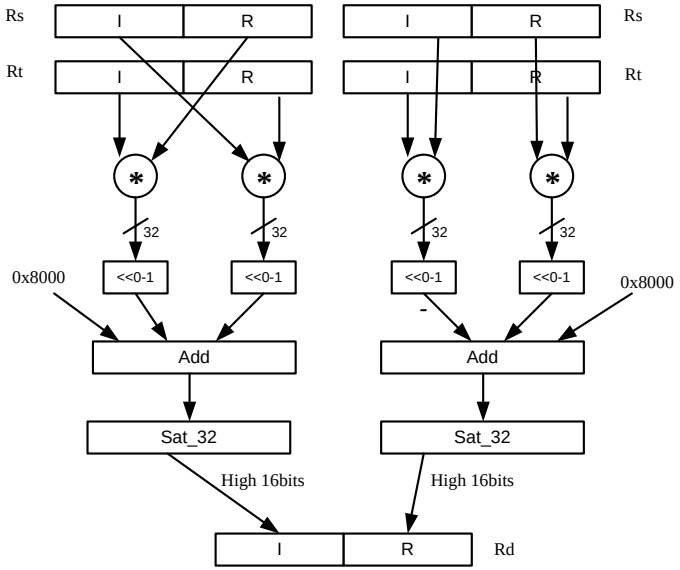
## What is VLIW? Very Long Instruction Word architecture

- Hardware designed to execute multiple instructions in parallel
- Instructions to be executed are statically scheduled by compiler
- Example VLIW instruction “packet”, performs 29 RISC-style operations in one cycle:

64-bit Load and
64-bit Store with post-update addressing

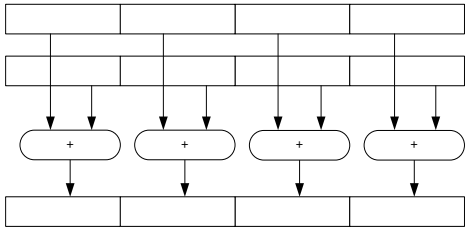
```
{ R17:16 = MEMD(R0++M1)
  MEMD(R6++M1) = R25:24
  R20 = CMPY(R20, R8):<<1:rnd:sat
  R11:10 = VADDH(R11:10, R13:12)
 }:endloop0
```

Complex Multiply




HW-loop end
- Dec count
- Compare
- Jump top

Vector 4x16-bit Add







QuIC QUALCOMM INNOVATION CENTER, INC.

# Motivation

- Qualcomm's Hexagon™ is a VLIW DSP with an optimizing assembler
  - llvm-mc adapted for use as a stand-alone assembler
- Very little native support for VLIW in the MC layer
  - Designed to stream a single MCInst at a time, with minimal modification
- Packet awareness needed for packet optimization passes, such as:
  - Enforcing instruction ordering within the packet
    - Packet must be shuffled to conform to hardware execution slot restrictions
  - Creation of duplex encodings for code size reduction
    - Combine two instructions as a single 32-bit word when possible
  - Handling new-value loads/stores
    - Use values loaded in the packet for other operations within the same packet
      - » Example:

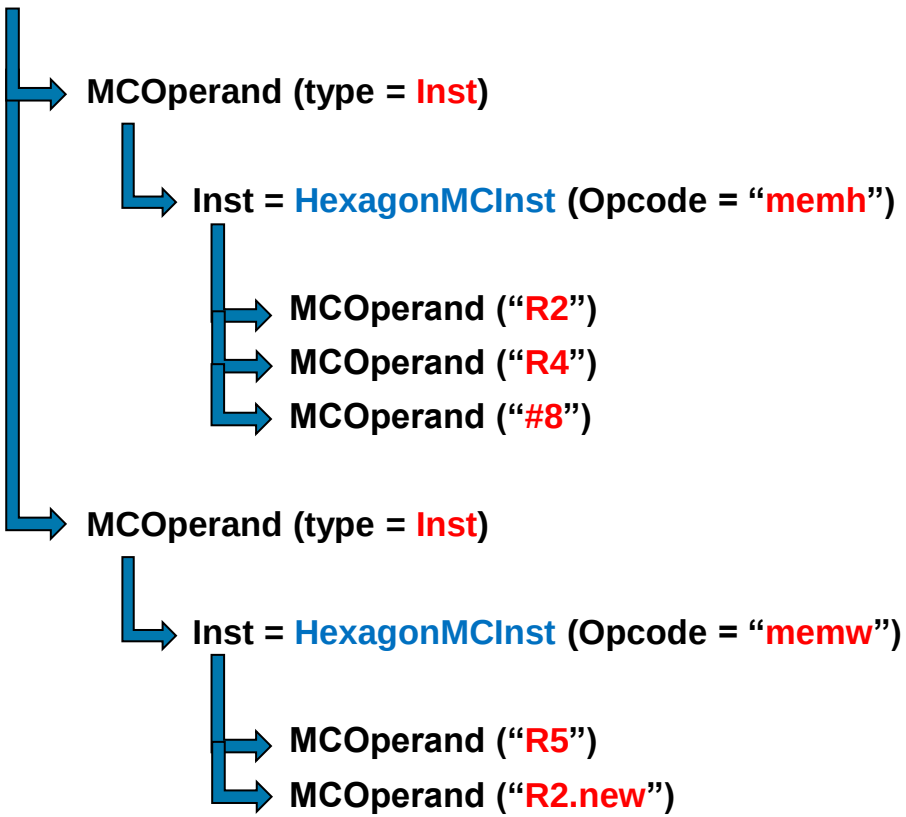
```
{ R2 = memh(R4+#8)     // load half-word
  memw(R5) = R2.new }  // store newly loaded value
```

# Implementation

- Extend MCInst via sub-class named “HexagonMCInst”
  - Necessary for capturing extra information needed for packet optimization
- Create nested HexagonMCInst bundle for each instruction packet
  - Define top-level HexagonMCInst with opcode = BUNDLE
  - Define individual HexagonMCInst packet instructions
  - Add all packet instructions to top-level bundle as instruction operands
- HexagonMCInst bundle passes through the MC layer intact
  - Treated as a single MCInst by the MC layer
  - Packet optimizations are applied to entire bundle in Hexagon-specific passes
  - Hexagon sub-classes of streamer objects unroll each bundle for final output

# Example

- Sample packet:
```
{ R2 = memh(R4+#8)     // load half-word
  memw(R5) = R2.new }  // store newly loaded value
```
- MC layer bundle :






QuIC QUALCOMM INNOVATION CENTER, INC.

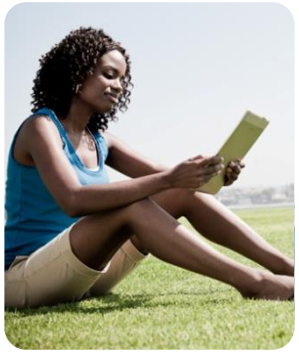
# Conclusion

- Basic design of MCInst is very well suited to VLIW
  - Bundling approach outlined in this presentation worked very well for representing instruction packets

- Other aspects of current MC layer design not well suited to VLIW
  - Parser assumes mnemonic is always the first token
    - Not true for Hexagon - more flexibility in mnemonic placement is needed
  - Parser does not support creation of packets
    - Hexagon parser requires lots of custom code in order to build instruction bundles
    - Need ability to make several token parsing passes prior to invoking streamer
  - Info needed for enforcing order in a packet is lost in lowering from MI to MC
    - Primary reason for sub-classing MCInst
    - Sub-classing raised other issues not covered here

- Future MC layer design decisions should take VLIW into consideration

# Thank You



©2013 Qualcomm Innovation Center, Inc.
Qualcomm and Hexagon are trademarks of QUALCOMM Incorporated, registered in the United States and other countries. All QUALCOMM Incorporated trademarks are used with permission. Other product and brand names may be trademarks or registered trademarks of their respective owners.

QuIC
QUALCOMM INNOVATION CENTER, INC.