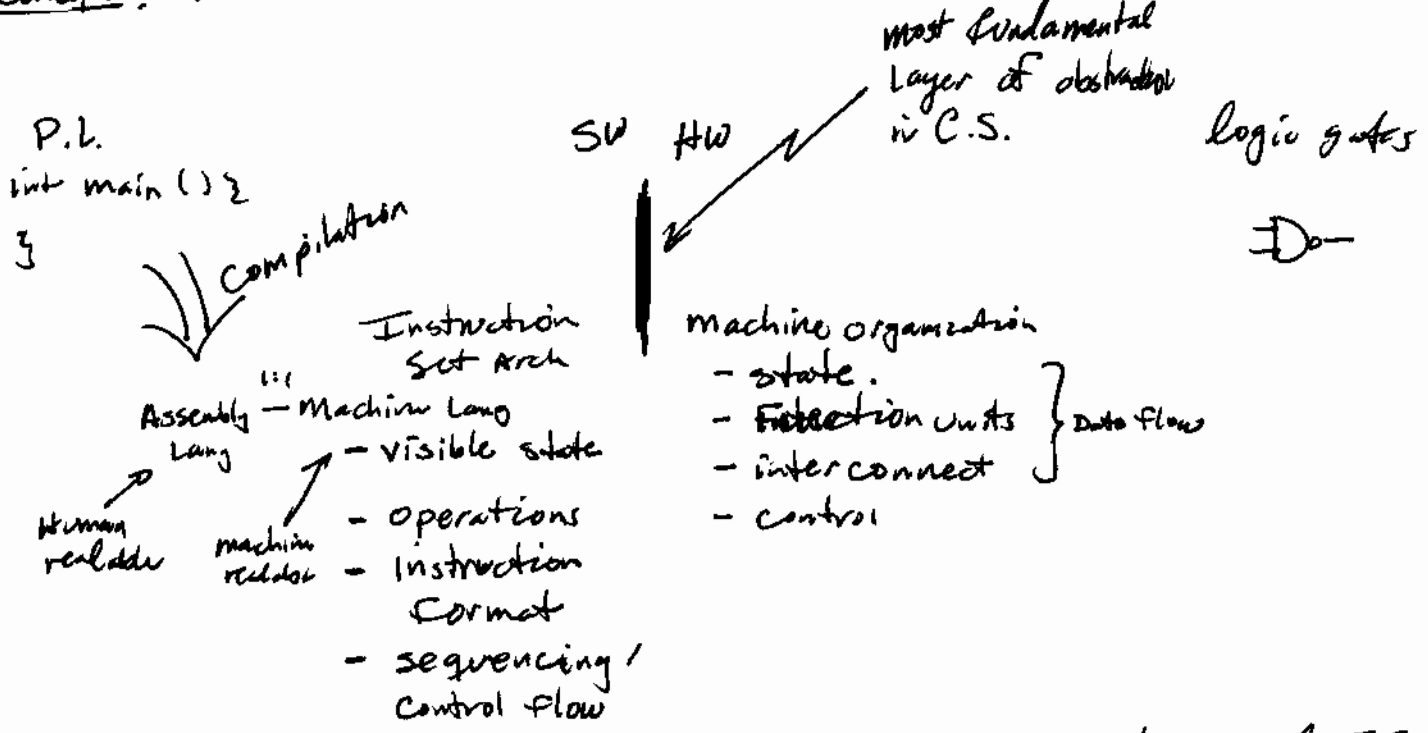
From Page No.

Outline
- Inst. Set Arch concept
- MIPS ISA
- Reg / Reg
  Reg / Ime
  mem & & Address
  Jmp & Branch

Concept: ISA



- Why it is important ?
  + design, test, or build HW
  + operating systems – device drivers, concurrency
  + embedded systems
  + performance tuning & analysis
  + compilers

- Classes of I.S.A.
  - MIPS is "Load/store"

- Brief history
  50's feature
  60's families
  70's CISC
  80's RISC
  + GPUs, VMs, ...

To Page No.

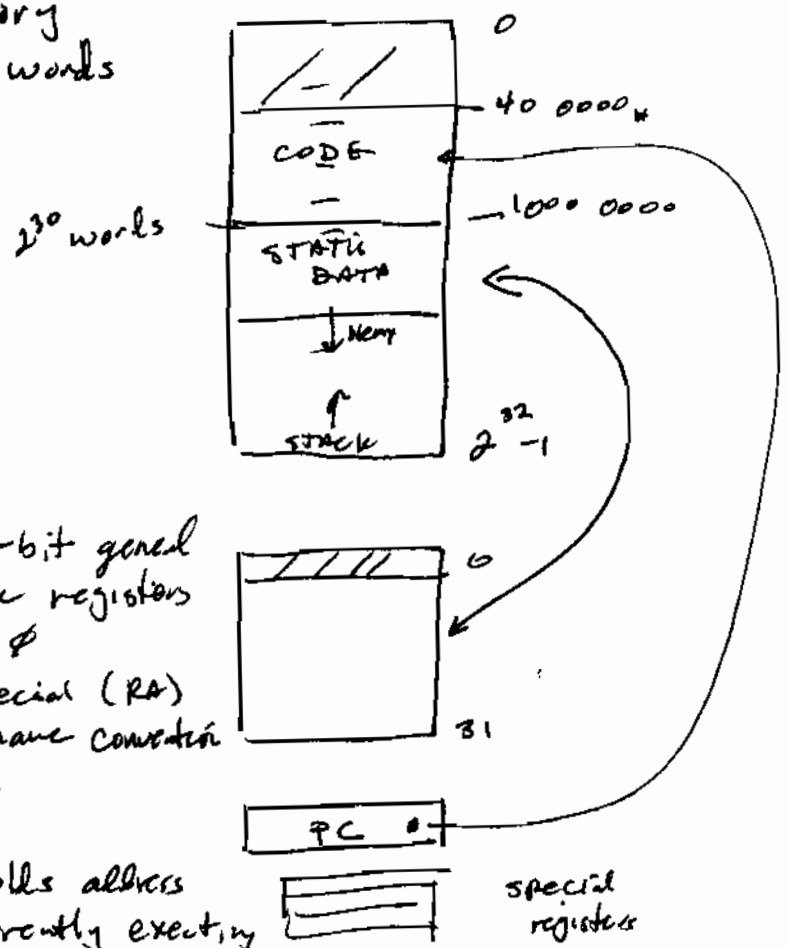
Witnessed & Understood by me,	Date	Invented by	Date
		Recorded by	

From Page No.__

MIPS

(1) Arch. State

- 32-bit byte addressable memory
- 32-bit words



- 32 32-bit general purpose registers
  $0 == 0
  $31 special (RA) reg have convention usage

- PC holds address of currently executing instruction

special registers

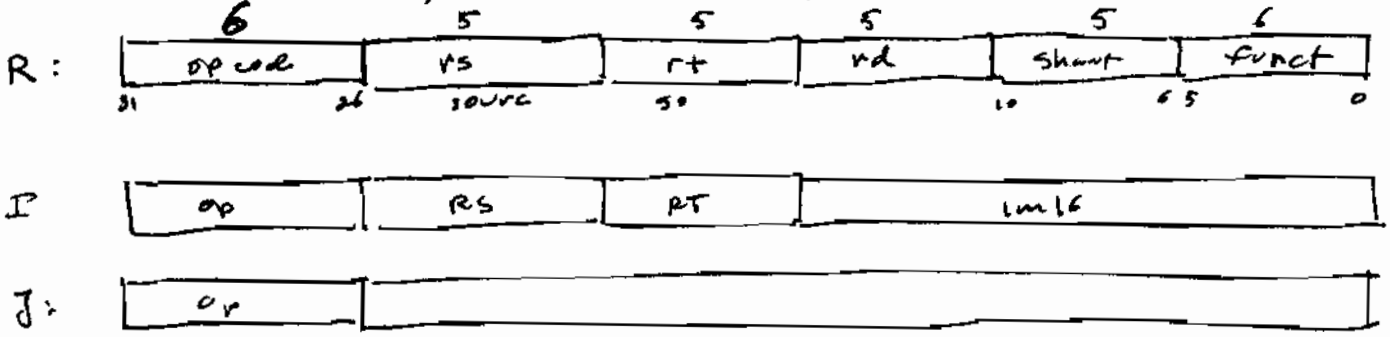
2. Operations
   - LOAD / store - move information to/from reg
     - registers are words; lb, lh throw parts away
     - 32 bit values are untyped
       - can be anything, meaning is what you do with th
   - Register/Reg or Reg/Immediate
     - Arithmetic, logical
   - Jumps - direct & indirect (absolute)
   - Branch on condition (relative)

To Page No.__

Witnessed & Understood by me,	Date	Invented by	Date
		Recorded by	

From Page No. ___

- Instruction Format - "Fixed Format" vs x86
  - all 32-bits, fields in "same place"



Instruction cycle:

Reg. Transfer

- fetch Inst — Inst ← Mem[PC]
- decode (dispatch on op) & funct — op?
- operand fetch - Register — R: R[rs], R[rt]; I
- Arith / Logic | Effective Add
- — Mem
- Register Write
- Update PC — PC ← PC + 4; Jm PC ← Addr; Br PC ← PC + cond ? off : 4

Direct

Reg - Reg Inst. (op == 0)
ADD, SUB, AND, OR, NOR, XOR
SLT,
Sll, Srl, Sra, Sllv

R[rd] ← R[rs] fun R[rt]
PC ← PC + 4

R[rd] ← R[rt] << shamt
>>

Reg - Immed
ADDI, ANDI, ORI, XORI, LUI
ADDIU
SLTI, SLTIU

R[rt] ← R[rs] op Im16

To Page No. ___

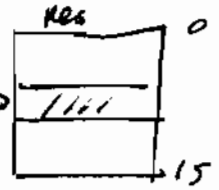
Witnessed & Understood by me,	Date	Invented by	Date
		Recorded by	

From Page No.

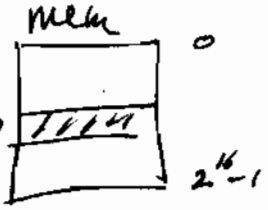
Operands in memory
- Load / store
- How are they located?
  - Effective Address calculation on register & immediate

Immediate op RS RT Imm

Register Direct op RS RT ADDR

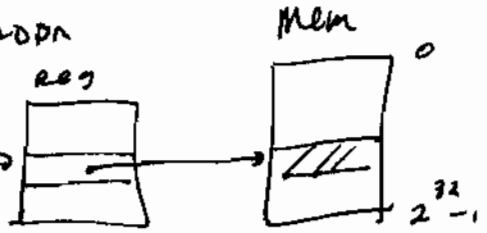


~~Register~~ L/S
Absolute op RS RT ~~ADDR~~



Register Indirect (~~Base~~)

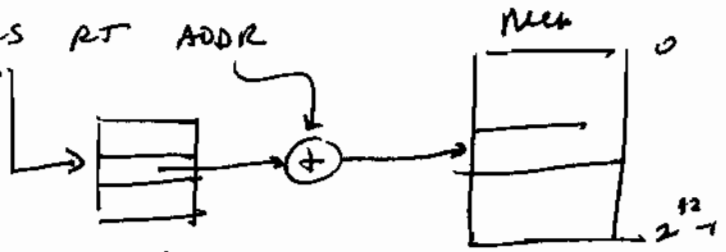
op RS RT ADDR



- pointers
- ARRAYS
- fully gene

Base + offset

op RS RT ADDR



optimization
- structs
- stack fra

Many other options on x86
- memory indirect
- auto increment, autodecrement

Mips does them all with Base + offset - either zero

To Page No.

Witnessed & Understood by me,	Date	Invented by	Date
		Recorded by	

From Page No.

LW Rt, Im16 (RS) — RT ← Mem [Reg (RS) + Im16]
LH, LHU
LB, LBU

SW Rt, Im16 (RS) — Mem [Reg(RS) + Im16] ← Reg [rs]

Control Transfer

Unconditional

J ADDR } Absolute or Direct — PC ← Addr
JAL ADDR } — R[31] ← PC+4, PC ← Addr

↳ unconditional

JR rs — indirect jump — PC ← R[rs]
- Return
- dispatch
- function ptrs

conditional

BEQ RS, Rt, Addr16 — if R[rs] == R[RT] PC ← PC+4+ADDR16
BNE RS, Rt Addr16 — else PC ← PC+4

Pseudo inst
BLT, BGT, BLE, BGTE, LI, MOVE

~~Coprocessor~~
Other arithmetic
MULT, DIV, FP

other system inst

To Page No.

Witnessed & Understood by me,	Date	Invented by	Date
		Recorded by	