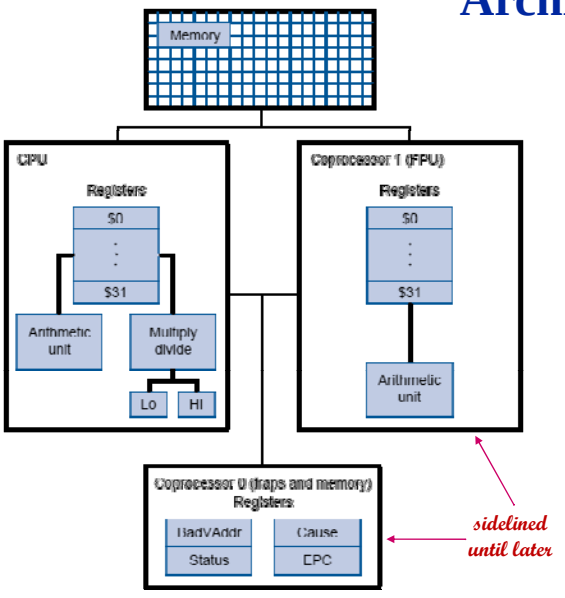
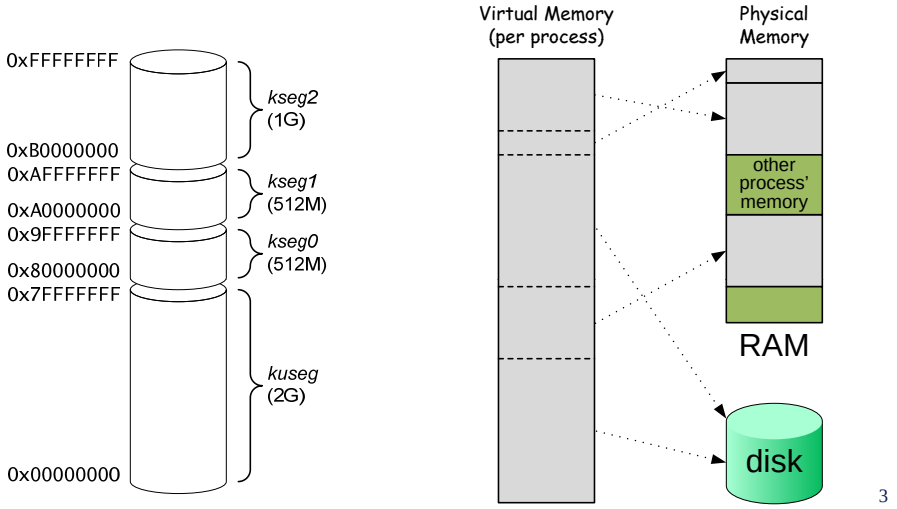
MIPS Architectures
(quick historical perspective)

- Primarily grew out of research done at Stanford
- Evolved over time from MIPS I through MIPS V
- Late 1990's: designs formalized to 2 basic architectures
  - 32-bit MIPS32 and 64-bit MIPS64 architectures
- Mid-1980's: R2000 announced
  - 1st MIPS processor implementation (32-bit)
- 3 years later: improved version R3000 became available
- Early 1990's: R4000 released
  - 1st 64-bit implementation
- Our focus: R2000/R3000
  - (sufficient for us to learn MIPS assembly language)

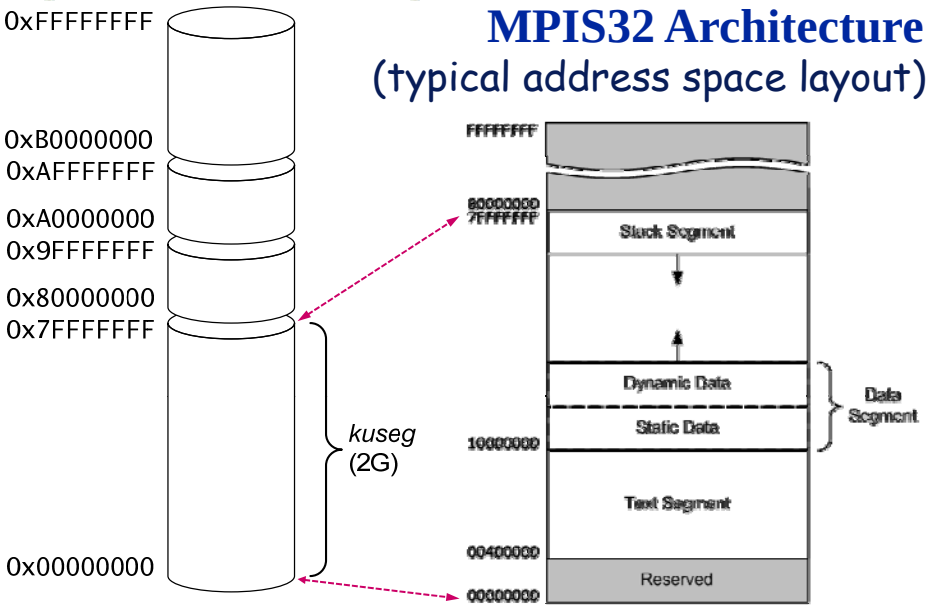
MIPS32 Architecture



MPIS32 Architecture
(virtual memory & address space)



MPIS32 Architecture
(typical address space layout)



MIPS32 Architecture
(notes on typical address space layout)

- Divided into 3 parts
  - 1st part: ***text segment*** → holds program's instructions
  - 2nd part: ***data segment*** → further divided into 2 parts
    - *Static data* → size known at compile time and lifetime is *static*
      (NB: static lifetime → can be accessed during program's entire execution)
    - *Dynamic data* → allocated by program at runtime
  - 3rd part: ***stack segment*** → program's runtime stack
- "3-part memory division" setup
  - Not the only way possible
  - Has 2 nice properties for the 2 dynamically expandable segments
    - They are as far apart (from each other) as possible
    - They can grow (in "opposing" directions) to use program's entire address space

MIPS32 Architecture
(CPU registers)

- 32 32-bit general purpose registers (GPR's)
  - Can reference by number: $0, $1, …, $31
  - Or by name (more meaningful, so better way to go): $zero, $at, …, $ra
  - (NOTE: each number/name must be preceded by $)
- 2 32-bit special purpose registers
  - Hi, Lo
- Not all 32 GPR's are "truly" general purpose
  - $zero ($0) is *hardwired to zero*
    - For use when 0 is needed as *source* operand
    - Will have no effect if used as *destination* operand
  - $ra ($31) serves specific purpose of *link register*
    - Return address automatically stored in it during procedure call
- Hi and Lo hold results of integer multiply and divide instructions
  - Multiply: higer-order 32 bits in Hi, lower-order 32 bits in Lo
  - Divide: remainder in Hi, quotient in Lo
  - (NOTE: Hi and Lo are accessed only *indirectly* through certain instructions)

MIPS32 Architecture
(usage convention for CPU registers)

Register Name(s)	Register Number(s)	Intended Use (by convention)	Preserved Across Call?
$zero	0	constant value 0	(N.A.)
$at	1	reserved for assembler	no
$v0, $v1	2, 3	result(s) of procedure	no
$a0, ..., $a3	4, ..., 7	argument(s) of procedure	no
$t0, ..., $t7	8, ..., 15	temporaries	no
$s0, ..., $s7	16, ..., 23	saved temporaries	yes
$t8, $t9	24, 25	temporaries	no
$k0, $k1	26, 27	reserved for OS kernel	no
$gp	28	global pointer	yes
$sp	29	stack pointer	yes
$fp	30	frame pointer	yes
$ra	31	return address	yes

MIPS32 Architecture
(more about CPU register usage convention)

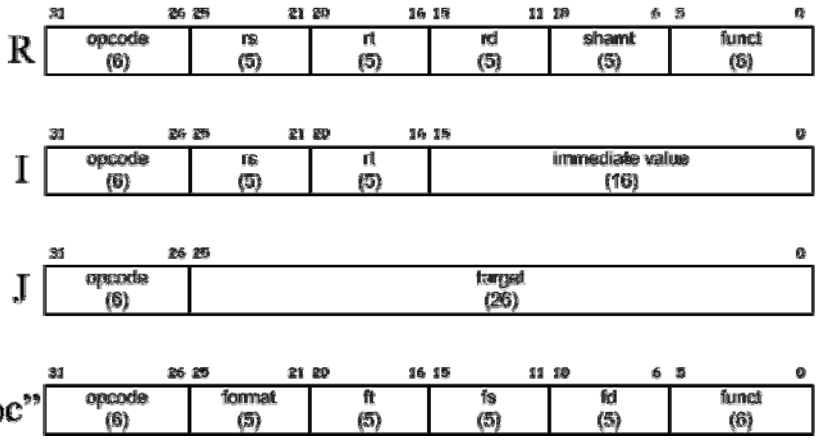
- Convention not required/enforced by hardware
  - If not followed in practice: code interoperability suffers
  - If not followed in this class: lose points
- Some notes on usage convention
  - $at ($1): not for use by programmer (**a**ssembler **t**emporary)
  - $v0, $v1 ($2, $3): for returning values from procedure (v → **v**alue?)
  - $a0, …, $a3 ($4, …, $7): for passing 1st 4 arguments to procedure (a → **a**rg?)
  - $t0, …, $t9 ($8, …, $15, $24, $25): *caller-saved* temporaries (t → **t**emp)
  - $s0, …, $s7 ($16, …, $23): *callee-saved* saved registers (s → **s**aved)
  - $k1, $k2 ($26, $27): not for use by programmer (k → **k**ernel)
  - $gp ($28): typically a pointer dedicated to MIPS system (gp → **g**lobal **p**ointer)
    - Points to static data segment: facilitates faster access to memory at 10000000h - 10010000h
  - $sp ($29): will see its use when doing procedure (sp → **s**tack **p**ointer)
  - $fp ($30): will see its use when doing procedure (fp → **f**rame **p**ointer)

MIPS32 Architecture
(CPU Register Usage Guide)

- In the early going (before getting into stack & doing functions):
  - $0 (or $zero): use wherever appropriate
  - $v0: use for when doing syscall (receiving value returned)
  - $v1: use as additional temporary (like $t)
  - $a0, $a1: use for when doing syscall (passing value(s) as input)
  - $a3, $a4: use as additional temporaries (like $t)
  - $t0, …, $t9: key temporaries

  - DON'T use the following:
    - $at
    - $s0, …, $s7
    - $k1, $k2
    - $gp
    - $sp
    - $ra

MPIS32 Architecture
(instruction formats)



MIPS32 Architecture
(more about instruction formats)

- R-type (R → **R**egister)
  - Instructions that don't require immediate value, target offset, memory address displacement or memory address to specify an operand, including:
    - All arithmetic and logic instructions with all operands in registers
    - Shift instructions
    - Register direct jump instructions
- I-type (I → **I**mmediate)
  - Instructions with an immediate operand
  - Branch instructions
  - Load and store instructions (including coprocessor load and store instructions)
- J-type (J → **J**ump)
  - 2 direct jump instructions
- "Coproc" (Coproc → **Coproc**essor) – self-coined type name
  - Coprocessor instructions