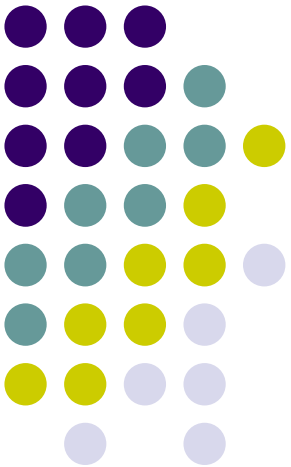
ECE/ME/EMA/CS 759
High Performance Computing for Engineering Applications

CMake

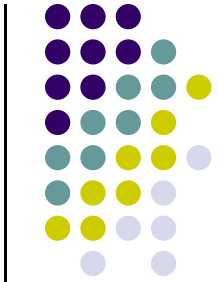
Registers

Pipelining in Sequential Computing



September 9, 2015

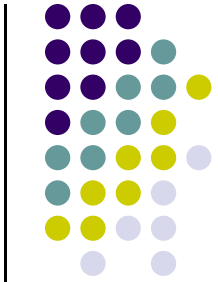
Quote of the Day



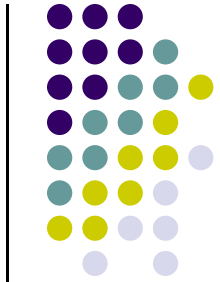
“Management is doing the things right. Leadership is doing the right things.”

Peter Drucker [1909-2005]

Before We Get Started



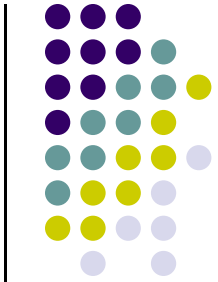
- Issues covered last time:
  - From a line of code to machine instructions
  - An instruction; i.e., a sequence of 1 and 0 bits, encapsulates a work order
  - Transistors can be organized to perform tasks that combine to fulfill a work order
  - Moore's Law has been guaranteeing a steady increase in the number of transistors we can pack per unit area
- Today's menu
  - CMake (Hammad)
  - Registers
  - Pipelining
- Assignment:
  - HW01 - due today: drop in the Learn@UW box
  - HW02 – assigned today (available today on the class website)
    - Due next Wd at 11:59 PM, use dropbox at Learn@UW

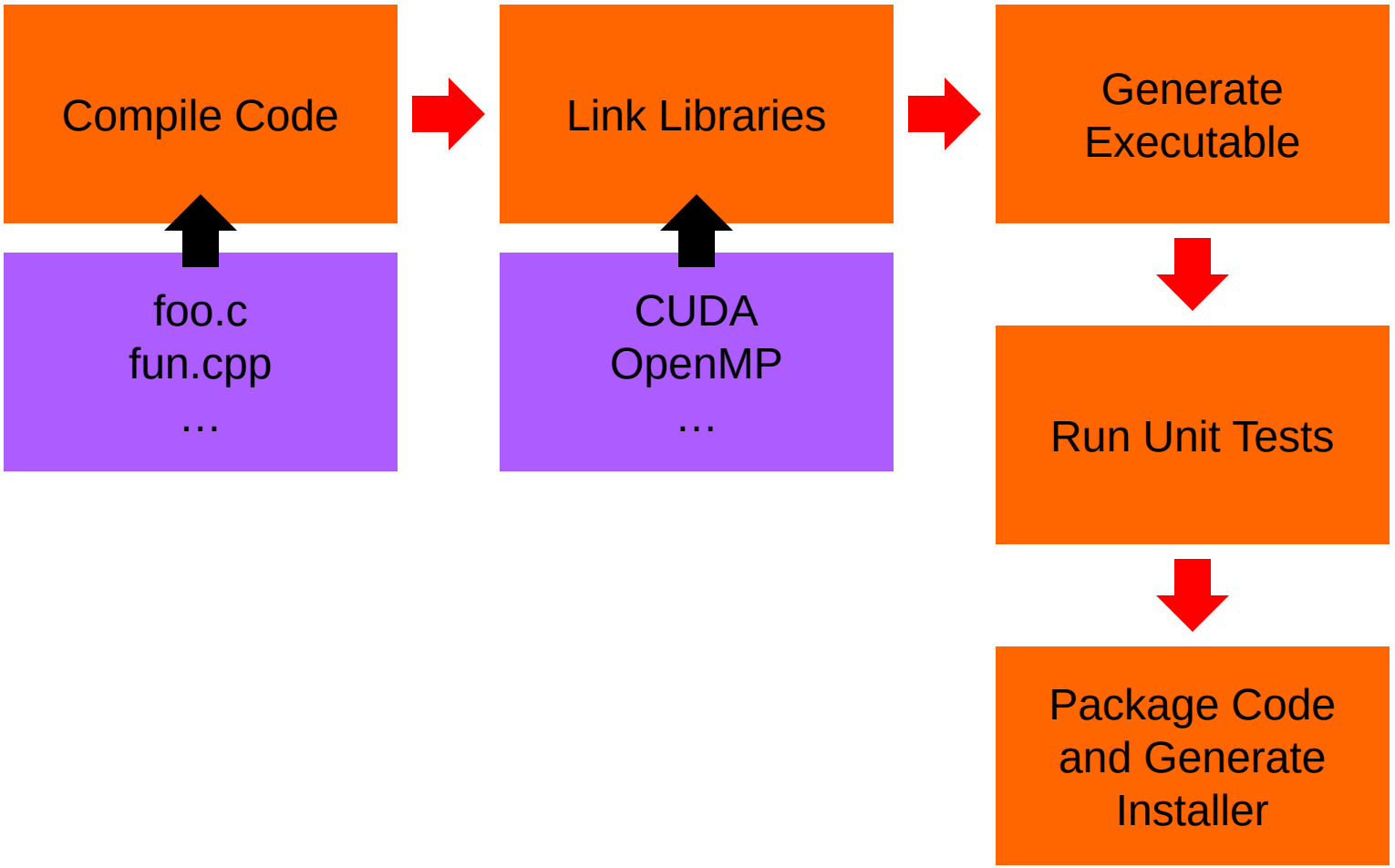


CMake

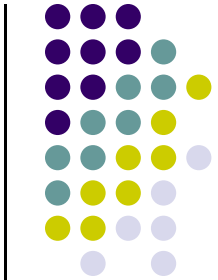
~ A Build System for Build Systems ~

Role of Build Systems



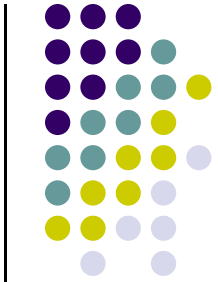


A Few Build Systems…



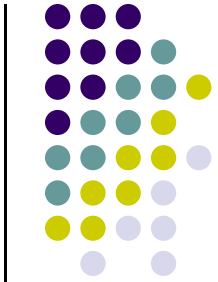
- (hand-written) makefiles
  - Portability depends on author
- Autotools (GNU build system)
  - Most familiar: ./configure && make && make install
  - But there's more: aclocal, autoheader, automake, autoconf,…
  - Require Cygwin or MSYS for Windows
- Eclipse, Visual Studio
  - Tied to IDE
  - Complex setup for large projects

…and Two More



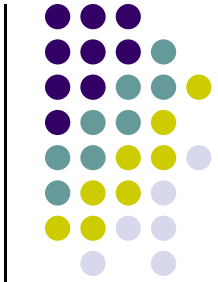
- SCons
  - Builds defined as Python scripts
  - Used by Blender, Doom3, NumPy, SciPy
- CMake
  - Can generate Eclipse projects, Visual Studio solutions, Makefiles, XCode projects, etc.
  - What we will use for the rest of ME759

Intro to CMake



- Projects are defined in simple text files
  - No more digging through stacks of config dialogs
  - Easy to diff
  - Easy to maintain under revision control (SVN, Mercurial, Git, etc.)
  - Works on any platform (Linux, Windows, OSX)
- User-configurable options set in the ccmake/cmake-gui programs
- Once configured, project files are generated for your system's native build environment (Eclipse, Visual Studio, Makefiles, Xcode, etc)

CMake Lingo



- CMakeLists.txt
  - Text file in which you define the bulk of the project
- Generator
  - Converts CMakeLists.txt to a project file for your IDE
- Cache
  - Stores environment-specific and user-configurable options
- Build type
  - Set of compiler/linker options
  - Some predefined setups:
    - debug, release, release with debug symbols, space-optimized release

CMake Configuration Options

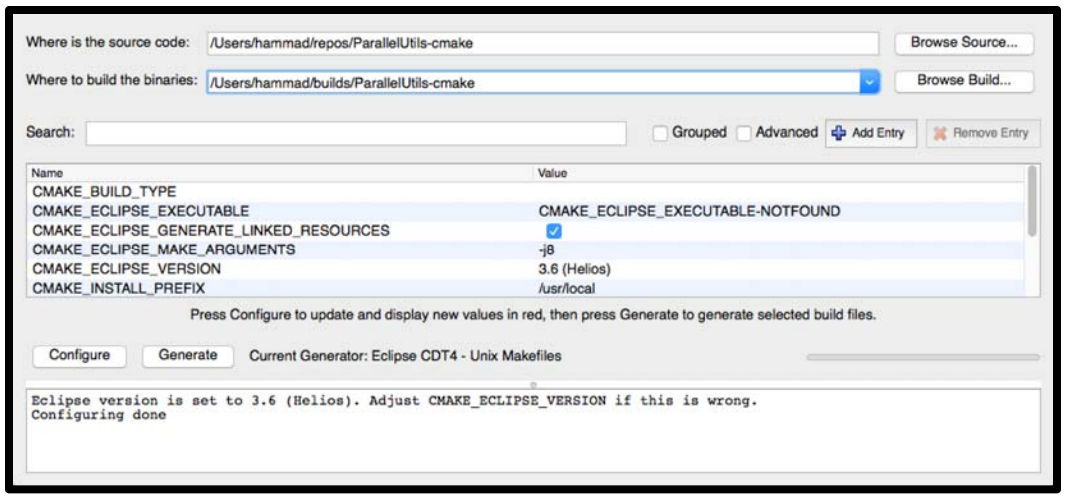
- “cmake”

```
→  ParallelUtils-cmake  cmake ~/repos/ParallelUtils-cmake
-- The C compiler identification is AppleClang 6.1.0.6020053
-- The CXX compiler identification is AppleClang 6.1.0.6020053
-- Check for working C compiler: /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/cc
-- Check for working C compiler: /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/c++
-- Check for working CXX compiler: /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /Users/hammad/builds/ParallelUtils-cmake
→  ParallelUtils-cmake
```

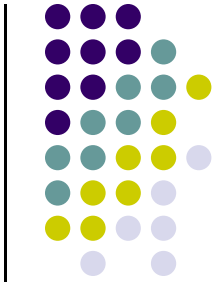
- “ccmake”

```
                                Page 1 of 1
CMAKE_BUILD_TYPE                *
CMAKE_INSTALL_PREFIX            */usr/local
CMAKE_OSX_ARCHITECTURES         *
CMAKE_OSX_DEPLOYMENT_TARGET     *
CMAKE_OSX_SYSROOT               *

CMAKE_BUILD_TYPE: Choose the type of build, options are: None(CMAKE_CXX_FLAGS or CMAKE_C_FLAGS used) Debug Release RelWithDebInfo MinSizeRel.
Press [enter] to edit option                                  CMake Version 3.3.1
Press [c] to configure
Press [h] for help            Press [q] to quit without generating
Press [t] to toggle advanced mode (Currently Off)
```

- “cmake-gui”



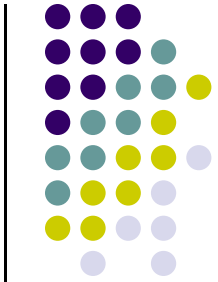
CMake Workflow



1. Write CMakeLists.txt
2. Select build directory in ccmake/cmake-gui
3. Choose generator for your environment
   - Eclipse, Visual Studio, Makefiles, etc
4. Configure project options,
   - Persistent, Saved in cache
5. Generate project files
6. Build project

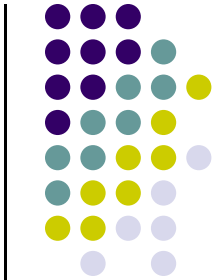
CMakeLists.txt

- Defines the entire project and build process
- Watch out: name must be **exactly** CMakeLists.txt
- Contents themselves are case insensitive
  - But **be consistent**
  - Commonly found in recent projects:
    - functions()
    - VARIABLES
- 20/80 rule: 20% of commands do 80% of what you'll need
- Documentation (CMake 2.8): http://cmake.org/cmake/help/cmake-2-8-docs.html



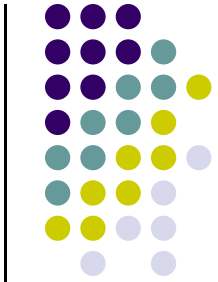
add_custom_command
add_custom_target
add_definitions
add_dependencies
add_executable
add_library
add_subdirectory
break
cmake_policy
configure_file
else
elseif
endforeach
endfunction
endif
endmacro
endwhile
execute_process
export
file
find_file
foreach
function
if
include
include_directories
install
link_directories
macro
message
option
project
return
set
string
target_link_libraries
while
add_custom_command

CMakeLists.txt: A Few Other Functions



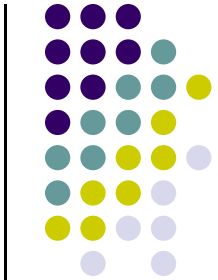
- configure_file: do find/replace on files
- ExternalProject: require an external project to be built before building your own
- find_package(foo): see if package foo is available on this system
  - This makes setting up CUDA and MPI relatively painless
  - But, FindFoo.cmake script must already be written
- math: perform arbitrary math operations
- {add,remove}_definitions: set/remove preprocessor definitions

Basic CMakeLists.txt



```
# Set the required version of CMake
cmake_minimum_required(VERSION 2.8)

# Set your project title
project(ME759)

# Look for CUDA and set up the build environment
# Flag ‘REQUIRED’ forces us to set up CUDA correctly before building
find_package(“CUDA” REQUIRED)

# Finally, we would like to create a program ‘foo’
# from the files ‘foo.cu’ and ‘bar.cu’
# Using cuda_add_executable tells CMake to use with nvcc instead of gcc
cuda_add_executable(foo foo.cu bar.cu)
```

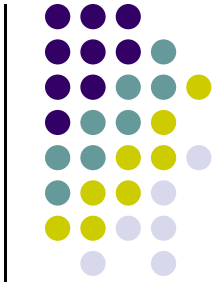
CMake for ME759



- A template available at https://github.com/uwsbel/ParallelUtils-cmake
- Has macros for CUDA, MPI, and OpenMP projects
  - To use:
    - Copy to your source directory
    - Uncomment relevant sections of CMakeLists.txt
    - Modify for your assignments

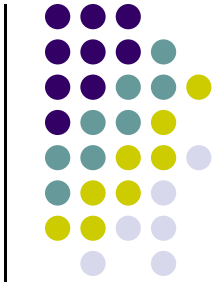
- Useful command: **`add_subdirectory`**
  - Allows you to have a single main CMakeLists.txt with assignment-specific ones in subdirs

CMakeLists.txt from Template



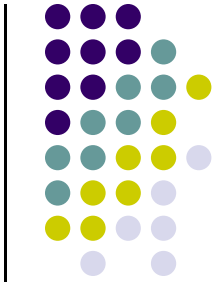
```
# Minimum version of CMake required. Don't touch.
cmake_minimum_required(VERSION 2.8)

# Set the name of your project
project(ME759)

# Include macros from the SBEL utils library
Include(ParallelUtils.cmake)

## Example CUDA program
enable_cuda_support()
cuda_add_executable(bandwidthTest bandwidthTest.cu)
```

What This Shows…



- Including commands from another file
- Running a macro (no arguments)
- Adding a CUDA executable to build
- **`ParallelUtils.cmake`** has more, see comments

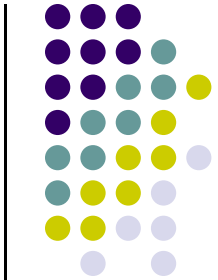
In-source v. Out-of-source Builds



- In-source builds
  - Binaries & project files generated alongside source code
  - Need to pay attention if using version control
  - IDEs (Eclipse) prefer this method
    - See http://www.cmake.org/Wiki/Eclipse_CDT4_Generator

- Out-of-source builds
  - Binaries & project files in separate directory
  - Easy to clean – just delete it
  - Only need to **`checkin/commit`** the source directory
  - This is the recommended way to build your code

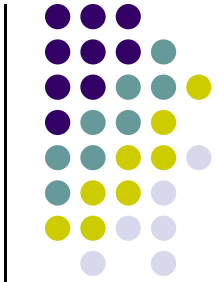
cmake-gui



- User-configurable options are set here

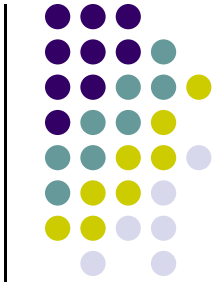
- Set source and build directories
  - Must decide between in-source v. out-of-source build
- New build dir/cleared cache: nothing there
  - Hit ‘Configure’ to select generator & start configuring
- New/changed options are shown in red
  - Modify if need be, then keep hitting configure until done
- ‘Generate’ creates the project files
- Feel free to venture into ‘Advanced’ options
  - Can manually set compiler/linker options here
  - Remember this: do a “**`File > Delete Cache`**” if something gets messed up

cmake-gui gotchas



- If you need a library/path/variable, make sure it is found
  - Will show up as **`{FOO}_NOT_FOUND`** in the config options
  - Can be manually set if need be
    - But you should probably first determine why it's not being done automatically
- Option not showing up? Hit Configure again, check advanced
- Strange issues? Clear the cache
  - Similar to **"`make distclean`"**

Using Projects, Compiling

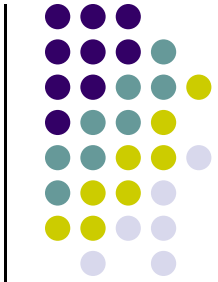


- After generating the project files, open in your IDE
  - Eclipse: **`File > Import Project`**
  - Visual Studio: open the solution
  - Makefiles/Eclipse: **`make`** (**`make -j4`** for parallel build w/ 4 threads)

- Source code should be in there, even if using out-of-source (linked to the source directory)

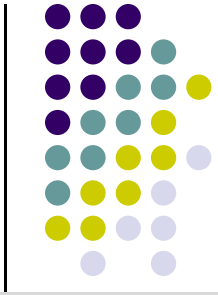
- **`CMake`** will automatically run when building to update project/make files
  - No need to open **`cmake-gui`** again unless changing options
  - Visual Studio may ask to reload the project; do it

Example Directory Structure



- me759_homework/
  - CMakeLists.txt — Main CMakeLists.txt
  - homework_01/
    - CMakeLists.txt — Homework Specific CMakeLists.txt
    - hw01.cpp
  - homework_02/
    - CMakeLists.txt — Homework Specific CMakeLists.txt
    - hw02.cu
  - …

Cmake File Example



Main CMakeLists.txt

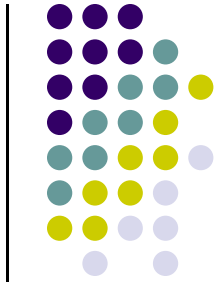
```
# Set the required version of CMake
cmake_minimum_required(VERSION 2.8)
# Set your project title
project(ME759_Homework)
# Include macros from the SBEL utils library
Include(ParallelUtils.cmake)
enable_cuda_support()

add_subdirectory(homework_01)
add_subdirectory(homework_02)
…
```

Homework Spefic CMakeLists.txt

```
add_executable(hw01 hw01.cpp)
…
```

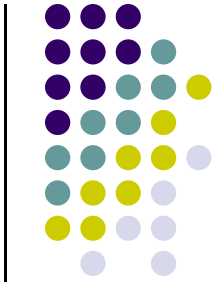
Homework Spefic CMakeLists.txt

```
cuda_add_executable(hw02 hw02.cu)
…
```



End Build Tools/Approaches
Go Back to Usual Program

Job Submission Option 1: Batch Mode



bandwidthTest.sh

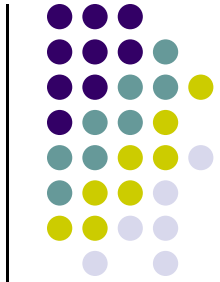
(you'll have to create this file)

```
#!/bin/bash                          → Shell script
#SBATCH -p slurm_me759               → Use Class Queue
#SBATCH –-job-name=bandwidthTest     → Name of job
#SBATCH –N 1 -n 1 –-gres=gpu:1       → Resource selection
#SBATCH -o bandwidthTest.o%j         → Set output file
cd $SLURM_SUBMIT_DIR                 → Set Work Directory
./bandwidthTest                      → Run!
```

Submit with:

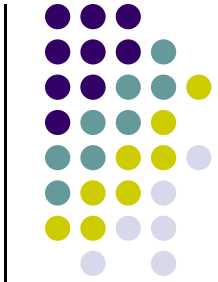
```
$ sbatch bandwidthTest.sh
```

Output placed in `bandwidthTest.o[0-9]*`



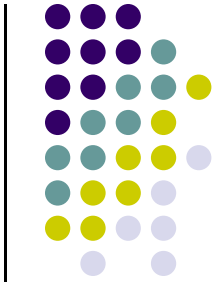
Registers

Registers



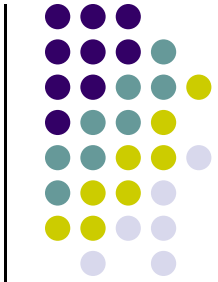
- Instruction cycle: fetch-decode-execute (FDX)
- CU – responsible for controlling/managing the process that eventually delivers the action/work order baked into an instruction
- ALU – busy worker that does +,-,*,AND,OR, etc. per the request of the CU
- The instruction that is being executed should be stored somewhere
- Fulfilling the action baked into an instruction usually involves handling input values and generating output values
  - This data needs to be stored somewhere

Registers



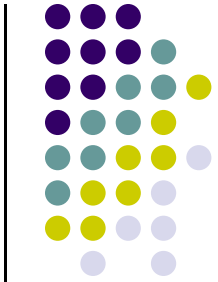
- Registers, quick facts:
  - A register is an entity whose role is that of storing information
    - Information comes in two flavors: DATA or MACHINE INSTRUCTION
  - A register is the type of storage with the shortest latency – it's closest to the ALU/CU
  - Typically, you; i.e., the programmer, cannot control what gets kept in registers
    - GPU computing is slightly odd
- Why are registers so special?
  - If you need information, there is no faster way to access it than when it's in a register

Registers



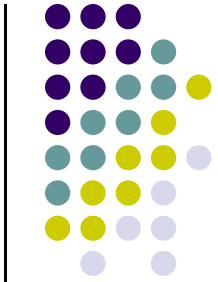
- The number AND size of registers used are specific to an ISA
  - The micro-architect needs to figure our how to physically construct the registers by squeezing them close to where they are used
    - The more registers, the better. Yet one cannot squeeze too many of them close to where the action happens (CU and ALU)
    - If they are not close enough to the CU and ALU, they shouldn't be called registers
      - Call them something else (like cache, for instance – a completely different animal)
  - In the MIPS; i.e., a RISC ISA: there are 32 registers of 32 bits and that's all, folks
  - In the x86; i.e., a CISC ISA: there are registers of various sizes and their number varies
    - Knights Corner x86 chip architecture used in Xeon Phi co-processors provisions for 512-bit wide SIMD registers
    - Most of the registers on the modern x86 chips are 64 bits wide

Register Types



- Mentioned below are only several register types typically encountered in a CPU (abbreviation in parenthesis)
  - List not comprehensive, showing only the more important ones
- Instruction register (IR) – a register that holds the instruction that is executed
  - Sometimes known as “current instruction register” CIR
- Program Counter (PC) – a register that holds the address of the next instruction that will be executed
  - NOTE: unlike IR, PC contains an *address* of an instruction, not the actual instruction

Register Types [Cntd.]

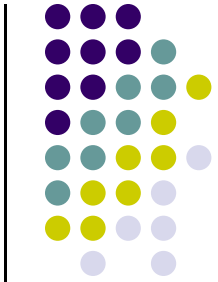


- Memory Data Register (MDR) – register that holds data that has been read in from memory or, alternatively, produced by the CPU and waiting to be stored in memory

- Memory Address Register (MAR) – the address of the memory location in memory (RAM) where input/output data is supposed to be read in/written out
  - NOTE: unlike MDR, MAR contains an *address* of a location in memory, not actual data

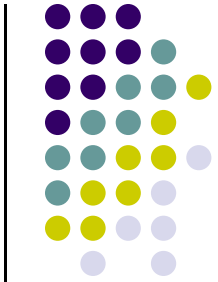
- Return Address (RA) – the address where upon finishing a sequence of instructions, the execution should jump and commence with the execution of subsequent instruction

Register Types [Cntd.]



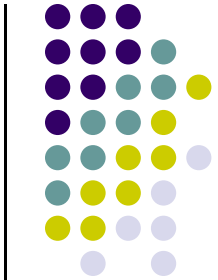
- Registers on previous two slides are a staple in most chip designs
- There are several other registers common to many chip designs yet they are encountered in different numbers
- Since they come in larger numbers they don't have an acronym
  - Example, MIPS
    - Registers for subroutine arguments (four of them) – a0 through a3
    - Registers for temporary variables (ten of them) – t0 through t9
    - Registers for saved temporary variables (eight of them) – s0 through s7
      - Saved between function calls

Register Types [Cntd.]

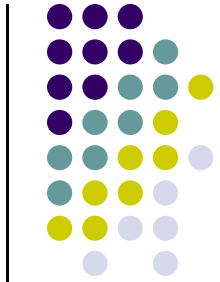


- Several other registers are involved in handling function calls
- Summarized below, but their meaning is only apparent in conjunction with the organization of the virtual memory
  - Global Pointer (gp) – a register that holds an address that points to the middle of a block of memory in the static data segment
  - Stack Pointer (sp) – a register that holds an address that points to the last location on the stack (top of the stack)
  - Frame Pointer (fp) - a register that holds an address that points to the beginning of the procedure frame (for instance, the previous **sp** before this function changed it's value)

Register, Departing Thoughts



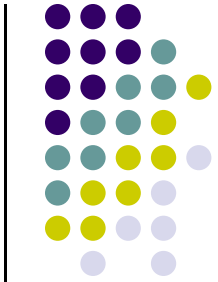
- Registers are precious resources, can't have enough of them
- Examples, what's out there:
  - In 32 bit MIPS ISA, there are 32 registers
  - On a GTX580 NVIDIA (Fermi architecture) card there are more than 500,000 32 bit temporary variable registers to keep busy 512 Scalar Processors (SPs) that make up 16 Stream Multiprocessors (SMs)
- Increasing the register count is not straightforward
  - Need to change the design of the chip; i.e., new hardware
  - Need to work out the control flow to leverage them; i.e., new logic



Pipelining

Charlie Chaplin - Modern Times (1936)

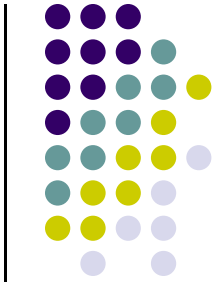
Pipelining, or the Assembly Line Concept



- Henry Ford: capitalized and improved the assembly line idea on an industrial scale and in the process shaped the automotive industry (Ford Model T)

- Vehicle assembly line: a good example of a pipelined process
  - Output of one stage (station) becomes input for the downstream stage (station)
  - A vehicle gets assembled after each cycle
    - The more cycles per hour, the more vehicles get manufactured
  - “cycle” is the time it takes from the moment a station gets its input to the moment it output leaves the station
  - It is bad if one station takes too long to produce its output since all the other stations idle for a while during each cycle of the production

Back to Execution of Machine Instructions



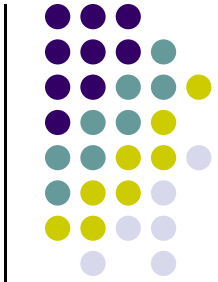
- FDX cycle: Fetch, Decode, Execute - carried out in conjunction with each instruction

- A closer look at what gets fetched (instructions and data) and then what happens upon execution leads to a generic five stage process associated with an instruction

- "generic": in a first order approximation, these five stages capture what's happening for any instruction (some instructions might not have all five stages):
  - Stage 1: Fetch an instruction
  - Stage 2: Decode the instruction
  - Stage 3: Data access
  - Stage 4: Execute the operation (Ex.: might be a request to calculate an address)
  - Stage 5: Write-back into register file

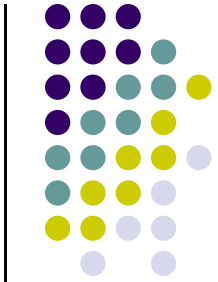
[Patterson, 4th edition]→

Pipelining, Basic Idea



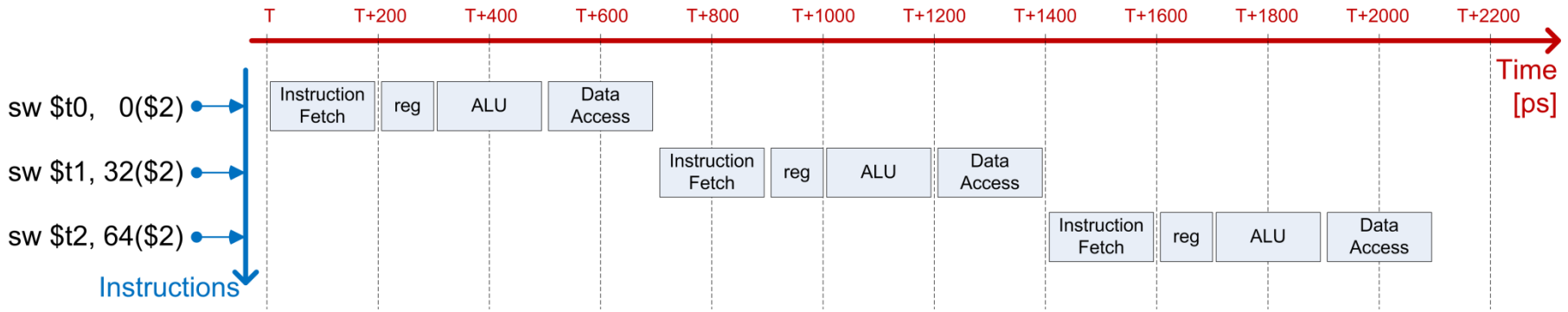
- At the cornerstone of pipelining is the observation that the following tasks can be worked upon simultaneously when processing five instructions:
  - Instruction 1 is in the 5th stage of the FDX cycle
  - Instruction 2 is in the 4th stage of the FDX cycle
  - Instruction 3 is in the 3rd stage of the FDX cycle
  - Instruction 4 is in the 2nd stage of the FDX cycle
  - Instruction 5 is in the 1st stage of the FDX cycle
- The above is a five stage pipeline
- An ideal situation is when each of these stages takes the same amount of time for completion
  - The pipeline is balanced
- If there is a stage that takes a significantly longer time since it does significantly more than the other stages, it should be broken into two and the length of the pipeline increases by one stage

Example: Streaming for execution 3 SW instructions

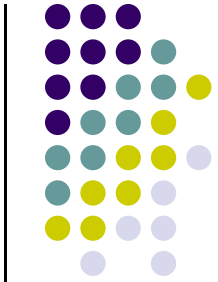


```
sw   $t0,  0($s2)
sw   $t1, 32($s2)
sw   $t2, 64($s2)
```

- Case 1: No pipelining – 2100 picoseconds [ps]

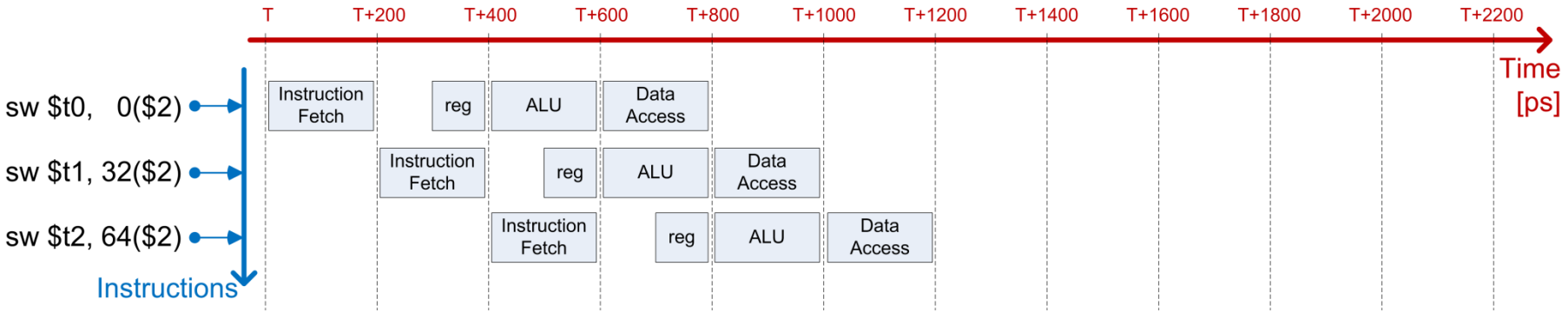


Example: Streaming for execution 3 SW instructions

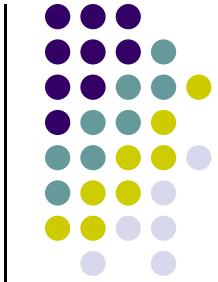


```
sw   $t0,  0($s2)
sw   $t1, 32($s2)
sw   $t2, 64($s2)
```

- Case 2: With pipelining – 1200 picoseconds [ps]



Pipelining, Benefits

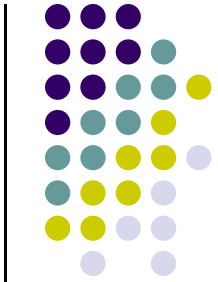


- Assume that you have
  1. A very large number of instructions
  2. Balanced stages
  3. A pipeline that is larger than or equal to the number “p” of stages associated with the typical ISA instruction

- If 1 through 3 above hold, in a first order approximation, the speed-up you get out of pipelining is approximately “p”

- Benefit stems from parallel processing of FDX stages
  - This kind of parallel processing of stages is transparent to the user
    - Unlike GPU or multicore parallel computing, user benefits for free

Pipelining, Good to Remember



- The amount of time required to complete one stage of the pipeline: one cycle
- Pipelined processor: one instruction processed in each cycle
- Nonpipelined processor: several cycles required to process an instruction:
  - Four cycles for SW, five for LW, four for add, etc.
- Important Remark:
  - Pipelining does not decrease the time to process one instruction but rather it increases the throughput of the processor by overlapping the execution of different stages of different instructions