UNIVERSITY COMPUTER SCIENCE • EXERCISE SHEET

3.6 Compilation and execution path

Name: _____ Class: _____ Date: _____ **Total: 24 marks**

KEY WORDS linker • preprocessor

Objective

Follow the pipeline; diagnose by stage.

1 Key knowledge and worked examples

■ **Four stages**

Preprocess (text; macro errors) → compile (types) → assemble (instructions) → link (symbols; undefined reference) ✓; the loader maps and starts (3.5).

■ **Worked example: the files**

main.c → main.i → main.s → main.o → prog – every intermediate can be kept and inspected ✓. Compiled = translate once, run many; interpreted = read and execute via a VM.

2 Practice

2.1 Which stage produces main.s? [3]

2.2 Which stage reports an undefined reference? [3]

2.3 Name the tool that turns .o files into an executable. [3]

3 Exam-style questions

3.1 Three failures, three stages.

(i) A macro used before definition – which stage, which error class? (ii) A function declared but never defined – stage and message? (iii) An invalid register name in inline assembly – stage? For each: name the stage, the tool, and why earlier stages did not catch it. [8]

3.2 Compare compiled and interpreted execution for the same source.

State when translation happens in each model and give one consequence for start-up speed and one for portability. Explain why separate compilation lets a million-line program rebuild in one file's time, and what the linker adds back. [7]

Solutions

2.1

- The compiler (cc1): main.i → main.s ✓.

2.2

- The linker (ld) ✓.

2.3

- The linker ✓.

3.1

- (i) Preprocessor (cpp): macro-expansion error – pure text, no type knowledge exists yet ✓.
- (ii) Linker (ld): undefined reference – the object exists; the symbol does not ✓.
- (iii) Assembler (as): instruction error – earlier stages see C, not register names ✓.
- Each stage owns one language: text, types, instructions, symbols – errors localise by construction ✓.

3.2

- Compiled: translated before running – faster start-up after the build, binary tied to the machine ✓.
- Interpreted: read and executed at run time – slower per line but portable to any machine with the VM ✓.
- Separate compilation: only changed files recompile; the linker re-resolves symbols across the objects to rebuild the whole ✓.